

(19) 日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11) 特許出願公開番号

特開平8-16429

(43) 公開日 平成8年(1996)1月19日

(51) Int.Cl. <sup>9</sup>	識別記号	庁内整理番号	F I	技術表示箇所
G 0 6 F 11/28	A	7313-5B		
9/06	5 4 0 S	7230-5B		
9/38	3 7 0 X			
15/16	4 3 0 A			
	4 5 0 Z			

審査請求 未請求 請求項の数50 F D (全 67 頁)

(21) 出願番号 特願平7-127577

(22) 出願日 平成7年(1995)4月28日

(31) 優先権主張番号 特願平6-114668

(32) 優先日 平6(1994)4月28日

(33) 優先権主張国 日本 (J P)

(71) 出願人 000003078

株式会社東芝

神奈川県川崎市幸区堀川町72番地

(72) 発明者 内平 直志

神奈川県川崎市幸区柳町70番地 株式会社

東芝柳町工場内

(72) 発明者 本位田 真一

神奈川県川崎市幸区柳町70番地 株式会社

東芝柳町工場内

(72) 発明者 大須賀 昭彦

神奈川県川崎市幸区柳町70番地 株式会社

東芝柳町工場内

(74) 代理人 弁理士 鈴江 武彦

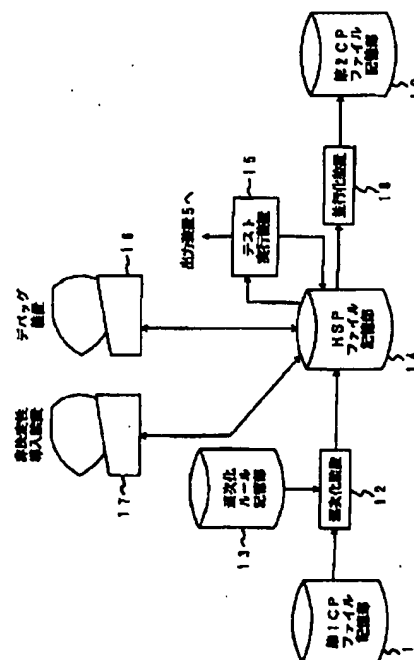
最終頁に続く

(54) 【発明の名称】 並行プログラム作成支援装置及び並行プログラム作成方法並びに並行プログラム実行装置

## (57) 【要約】

【目的】 並行プログラムのテスト・デバッグを容易に実現でき、並行プログラムの開発を効率よく行うことができる並行プログラムの作成方法及びその作成支援装置並びに効果的なテスト・デバッグ及び再現性を保証した部分並行実行を可能とする並行プログラム実行装置を提供すること。

【構成】 本発明の並行プログラム作成支援装置は、並行構造を有する第1並行プログラムを逐次実行可能な逐次プログラムに変換する逐次化手段12と、前記逐次プログラムのテスト・デバッグを行い、テスト・デバッグ情報を作成するテスト・デバッグ手段16と、テスト・デバッグ後の前記逐次プログラムを前記テスト・デバッグ情報に基づいて並行化することにより第2並行プログラムに変換する並行化手段18とを備えた。また、前記テスト・デバッグ手段は、前記逐次プログラムに対して並行性に関する情報を導入する手段を含む。



**【特許請求の範囲】**

【請求項1】 並行構造を有する第1並行プログラムを逐次実行可能な逐次プログラムに変換する逐次化手段と、前記逐次プログラムのテスト・デバッグを行い、テスト・デバッグ情報を作成するテスト・デバッグ手段と、テスト・デバッグ後の前記逐次プログラムを前記テスト・デバッグ情報に基づいて並行化することにより第2並行プログラムに変換する並行化手段と、を具備することを特徴とする並行プログラム作成支援装置。

【請求項2】 前記テスト・デバッグ手段は、前記逐次プログラムに対して並行性に関する情報を導入する手段を含むことを特徴とする請求項1記載の並行プログラム作成支援装置。

【請求項3】 前記逐次プログラムは、セクション情報と、プログラム構造情報と、逐次化情報とを含むことを特徴とする請求項1記載の並行プログラム作成支援装置。

【請求項4】 前記第1並行プログラムの並行構造を解析する解析手段を更に具備し、前記並行化手段は、前記テスト・デバッグ手段によりテスト・デバッグが行われた前記逐次プログラムを前記解析手段の解析結果を用いて並行化して、第2並行プログラムに変換する手段を含むことを特徴とする請求項1記載の並行プログラム作成支援装置。

【請求項5】 並行構造を有する第1並行プログラムの所定のセクション群を逐次実行可能な逐次プログラムに変換する逐次化手段と、

前記第1並行プログラムの所定のセクション群の並行構造を解析する並行構造解析手段と、

前記逐次プログラムの所定のセクション群の逐次構造を解析する逐次構造解析手段と、

前記並行構造解析手段で解析された並行構造に関する相互関係及び前記逐次構造解析手段で解析された逐次構造に関する相互関係をグラフ情報として表示するグラフ情報表示手段と、

前記逐次プログラムのうち選択された所定のセクション群を並行化して部分的に逐次構造を有する部分逐次プログラムに変換する部分逐次化手段と、

前記部分逐次プログラムを並行化して第2並行プログラムに変換する並行化手段と、を具備することを特徴とする並行プログラム作成支援装置。

【請求項6】 前記逐次化手段は、

前記第1並行プログラムを実行する実行手段と、

前記実行手段による実行ログを保存する保存手段と、

前記保存手段により保存された実行ログ及び前記第1並行プログラムを解析する解析手段と、

前記保存手段により保存された実行ログを前記解析手段による解析結果に基づいて並べ替える並べ替え手段と、

を含むことを特徴とする請求項1ないし請求項5のいずれかに記載の並行プログラム作成支援装置。

【請求項7】 前記実行手段は、

前記第1並行プログラムを解析してセクション情報の抽出をおこなうプログラム解析手段と、

前記セクション情報を記憶するセクション情報記憶手段と、

前記セクション情報記憶手段に記憶されたセクション情報を変更するセクション情報変更手段と、

前記セクション情報記憶手段により記憶されたセクション情報に基づいて前記並行プログラムの実行を行なう並行プログラム実行手段と、を含むことを特徴とする請求項6記載の並行プログラム作成支援装置。

【請求項8】 前記導入手段は、

前記逐次プログラムのプロセスの流れを制約と遷移条件からなるフィールドに変換して該フィールドを表すフィールドデータを生成するフィールドデータ生成手段と、前記手段により生成されたフィールドデータで表されるフィールドをチューニングするためのチューニング手段と、

前記フィールドデータ生成手段により生成されたフィールドデータで表されるフィールドを表示する表示手段と、含むことを特徴とする請求項1ないし請求項5のいずれかに記載の並行プログラム作成支援装置。

【請求項9】 前記逐次化手段は、前記第1並行プログラムを複数の超逐次プログラムに変換する手段を含み、前記テスト・デバッグ手段は、前記複数の超逐次プログラムを独立にテスト・デバッグする手段を含むことを特徴とする請求項1ないし請求項4のいずれかに記載の並行プログラム作成支援装置。

【請求項10】 並行構造を有する第1並行プログラムの所定のプロセス群を所定の実行順序に従って逐次実行可能な逐次プログラムに変換する逐次化手段と、

前記逐次構造プログラムに対して並行化の候補となるプロセス群を指定するプロセス群指定手段と、

前記手段により指定されたプロセス群の実行順序を入れ換えて前記逐次プログラムを複数の並行模擬プログラムに変換する模擬並行プログラム変換手段と、

前記複数の並行模擬プログラムを部分的に逐次構造を有する1つの部分逐次プログラムに変換する部分逐次化手段と、

前記部分逐次プログラムを並行化して第2並行プログラムに変換する並行化手段とを具備することを特徴とする並行プログラム作成支援装置。

【請求項11】 逐次化ルールを格納する逐次化ルール格納手段と、

前記逐次化ルール格納手段に格納された逐次化ルールに従って並行構造を有する第1並行プログラムを逐次実行可能な逐次プログラムに変換する逐次化手段と、

前記逐次プログラムに対して並行性に関する情報を導入すべく前記逐次化ルール格納手段に格納された逐次化ルールを修正する逐次化ルール修正手段と、

前記逐次プログラムのテスト・デバッグを行うためのテスト・デバッグ手段と、

前記逐次化ルール修正手段により並行性に関する情報が導入されかつ前記テスト・デバッグ手段によりテスト・デバッグが行われた逐次プログラムを並行化して第2並行プログラムに変換する並行化手段と、を具備することを特徴とする並行プログラム作成支援装置。

【請求項12】 プロセス群がメッセージ情報を交換しながら並行して動作する実行環境に用いられる並行プログラムの作成を支援する並行プログラム作成支援装置において、

第1並行プログラムのプロセス群の実行履歴であるログ情報を逐次化ルールとして取得するログ情報取得手段と、

前記ログ情報取得手段により取得されたログ情報を記憶するログ情報記憶手段と、

前記ログ情報記憶手段に記憶されているログ情報を修正するためのログ情報修正手段と、

前記ログ情報記憶手段に記憶されているログ情報に基づいて前記プロセス群を逐次的に起動制御するプロセス群制御手段と、

前記ログ情報記憶手段に記憶されたログ情報を並行化して第2並行プログラムに変換する並行化手段と、を具備することを特徴とする並行プログラム作成支援装置。

【請求項13】 前記ログ情報修正手段は、前記ログ情報記憶手段に記憶されているログ情報を時系列に表示する表示手段と、

前記表示手段により時系列に表示されたログ情報内のデータの順序の入れ換えを指示するための入れ替え指示手段と、

前記入れ替え指示手段による指示に従って前記ログ情報記憶手段に記憶されているログ情報を書き換える書換手段と、を含むことを特徴とする請求項12記載の並行プログラム作成支援装置。

【請求項14】 前記ログ情報修正手段は、プロセス間の処理タイミングの非決定性を導入する非決定性導入手段を含むことを特徴とする請求項12記載の並行プログラム作成支援装置。

【請求項15】 プロセス群がメッセージ情報を交換しながら並行して動作できる実行環境で前記プロセス群が実行順序規定情報に従って動作する並行プログラムを作成支援する並行プログラム作成支援装置において、前記実行順序規定情報を分割する実行順序情報分割手段と、

前記実行順序情報分割手段によって分割された実行順序情報に基づいて前記プロセスを起動制御するプロセス制御手段と、を具備することを特徴とする並行プログラム作成支援装置。

【請求項16】 前記実行順序規定情報を分割するための基準を与える分割基準指定手段を保持する手段を更に

具備することを特徴とする請求項15記載の並行プログラム作成支援装置。

【請求項17】 前記プロセス制御手段は、前記メッセージ交換の実行履歴であるログ情報を実行順序規定情報として用いるとともに、前記分割基準指定手段における分割基準として前記メッセージ中の宛先プロセス情報を基準として用いることを特徴とする請求項15記載の並行プログラム作成支援装置。

【請求項18】 前記プロセス制御手段を各プロセス毎に保持する手段を具備することを特徴とする請求項17記載の並行プログラム作成支援装置。

【請求項19】 前記実行順序情報分割手段によって分割された実行順序情報をそれぞれ別々に保存する分割実行順序情報保存手段を更に具備し前記プロセス制御手段は該プロセスに対応する分割実行順序情報保存手段に保存されている分割実行順序情報に基づいてプロセスを起動制御する手段を含むことを特徴とする請求項18記載の並行プログラム作成支援装置。

【請求項20】 前記プロセス制御手段は、前記プロセス群の実行履歴であるログ情報を前記実行順序規定情報として用いることを特徴とする請求項15記載の並行プログラム作成支援装置。

【請求項21】 並行構造を有する第1並行プログラムを逐次的にテスト実行する逐次的テスト実行手段と、前記テスト実行手段によるテスト実行の結果、正常に実行終了したテスト実行結果のバグのない逐次的実行ログを蓄積する実行ログ蓄積手段と、

前記実行ログ蓄積手段により蓄積された逐次的実行ログを並行化して第2並行プログラムに変換する並行化手段と、を具備することを特徴とする並行プログラム作成支援装置。

【請求項22】 並行構造を有する第1並行プログラムをテスト実行するテスト実行手段と、

前記テスト実行手段によるテスト実行の結果バグのない実行ログを蓄積する第1の実行ログ蓄積手段と、

前記テスト実行手段によるテスト実行の結果バグのある実行ログを蓄積する第2の実行ログ蓄積手段と、

前記第2の実行ログ蓄積手段により蓄積された実行ログを修正し、修正された実行ログを前記第1の実行ログ蓄積手段に蓄積する修正手段と、

前記第1の実行ログ蓄積手段により蓄積された実行ログを並行化して第2並行プログラムに変換する並行化手段と、を具備することを特徴とする並行プログラム作成支援装置。

【請求項23】 並行構造を有する第1並行プログラムを逐次実行可能な逐次プログラムに変換する第1ステップと、

前記逐次プログラムのテスト・デバッグを行い、テスト・デバッグ情報を作成する第2ステップと、

テスト・デバッグ後の前記逐次プログラムを前記テスト

・デバッグ情報に基づいて並行化することにより第2並行プログラムに変換する第3ステップと、を具備することを特徴とする並行プログラム作成方法。

【請求項24】 前記第2ステップは、前記逐次プログラムに対して並行性に関する情報を導入するサブステップを含むことを特徴とする請求項23記載の並行プログラム作成方法。

【請求項25】 前記第1ステップは、前記第1並行プログラムの並行構造を解析するサブステップを含み、前記第3ステップは、テスト・デバッグ後の前記逐次プログラムを前記第1並行プログラムの並行構造を用いて並行化するサブステップとを含むことを特徴とする請求項23記載の並行プログラム作成方法。

【請求項26】 前記第2ステップは、前記逐次プログラムに対して指定された非決定性を前記並行性に関する情報として導入するサブステップを含むことを特徴とする請求項23ないし請求項25のいずれかに記載の並行プログラム作成方法。

【請求項27】 前記第1ステップは、並行構造を有する前記第1並行プログラムの所定のセクション群を逐次実行可能な逐次プログラムに変換する第1サブステップと、前記第1並行プログラムの所定のセクション群の並行構造を解析する第2サブステップと、前記逐次プログラムの所定のセクション群の逐次構造を解析する第3サブステップと、前記第2サブステップで解析された並行構造に関する相互関係及び前記第3サブステップで解析された逐次構造に関する相互関係をグラフ情報として表示する第4サブステップと、前記逐次プログラムのうち選択された所定のセクション群を並行化して部分的に逐次構造を有する部分逐次プログラムに変換する第5サブステップと、を含み、前記第3ステップは、前記部分逐次プログラムを並行化して第2並行プログラムに変換するサブステップを含むことを特徴とする請求項23記載の並行プログラム作成方法。

【請求項28】 並行構造を有する前記第1並行プログラムの所定のセクション群を逐次実行可能な逐次プログラムに変換する第1ステップと、前記第1並行プログラムの所定のセクション群の並行構造を解析する第2ステップと、前記逐次プログラムの所定のセクション群の逐次構造を解析する第3ステップと、前記第2ステップで解析された並行構造に関する相互関係及び前記第3ステップで解析された逐次構造に関する相互関係をグラフ情報として表示する第4ステップと、前記逐次プログラムのうち選択された所定のセクション群を並行化して部分的に逐次構造を有する部分逐次プログラムに変換する第5ステップと、

前記部分逐次プログラムを並行化して第2並行プログラムに変換する第6ステップと、を具備することを特徴とする並行プログラム作成方法。

【請求項29】 前記第1サブステップは、前記逐次プログラムの所望の実行結果が得られるまで当該逐次プログラムのテスト・デバッグを行うサブステップを含むことを特徴とする請求項28記載の並行プログラム作成方法。

【請求項30】 前記第4サブステップは、前記所定のセクション群をノードとし、前記並行構造に関する相互関係を第1アークとし、前記逐次構造に関する相互関係を第2アークとして前記グラフ情報を表示するサブステップを含むことを特徴とする請求項28記載の並行プログラム作成方法。

【請求項31】 前記第5サブステップは、前記部分逐次プログラムの所望の実行結果が得られるまで該部分逐次プログラムのテスト・デバッグを行うサブステップを含むことを特徴とする請求項28記載の並行プログラム作成方法。

【請求項32】 前記第5サブステップは、前記部分逐次プログラムにおける所定のセクション群の逐次構造を解析するサブステップを含み、前記第1ステップは、前記第4サブステップから第5サブステップを所定の回数繰り返すサブステップを更に含むことを特徴とする請求項28記載の並行プログラム作成方法。

【請求項33】 前記第1ステップは、前記第1並行プログラムを実行する第1サブステップと、前記第1サブステップによる実行ログを保存する第2サブステップと、前記第2サブステップにより保存された実行ログ及び前記第1並行プログラムを解析する第3サブステップと、前記第2サブステップにより保存された実行ログを前記第3サブステップによる解析結果に基づいて並べ替える第4サブステップと、を含むことを特徴とする請求項23ないし請求項25、請求項28又は請求項29のいずれかに記載の並行プログラム作成方法。

【請求項34】 前記第3サブステップは、前記第2サブステップにより保存された実行ログ及び前記第1並行プログラムからプロセスの先行関係を抽出し先行関係情報として保存するサブステップを含むことを特徴とする請求項33記載の並行プログラム作成方法。

【請求項35】 前記第2ステップは、前記逐次プログラムのプロセスの流れを制約と遷移条件からなるフィールドに変換するサブステップと、前記フィールドをチューニングするサブステップと、前記並行性に関する情報を導入するサブステップとを含むことを特徴とする請求項23ないし請求項25のいずれかに記載の並行プログラム作成方法。

【請求項36】複数のプロセス群を有し並行構造を有する第1並行プログラムの所定のプロセス群を所定の実行順序に従って逐次実行可能な逐次プログラムに変換する第1ステップと、

前記逐次プログラムに対して並行化の候補となるプロセス群を指定する第2ステップと、

前記第2ステップにより指定されたプロセス群の実行順序を入れ換えて前記逐次プログラムを複数の並行模擬プログラムに変換する第3ステップと、

前記複数の並行模擬プログラムを部分的に逐次構造を有する1つの部分逐次プログラムに変換する第4ステップと、

前記部分逐次プログラムを並行化して第2並行プログラムに変換する第5ステップと、を具備することを特徴とする並行プログラム作成方法。

【請求項37】 前記第1ステップは、前記逐次プログラムの所望の実行結果が得られるまで該逐次プログラムのテスト・デバッグを行うサブステップを含むことを特徴とする請求項36記載の並行プログラム作成方法。

【請求項38】 前記第2ステップは、前記第1並行プログラムを解析するサブステップと、前記解析結果から前記並行化の候補となるプロセス群を抽出するサブステップと、を含むことを特徴とする請求項36記載の並行プログラム作成方法。

【請求項39】 前記第3ステップは、前記複数の並行模擬プログラムの所望の実行結果が得られるまで前記複数の並行模擬プログラムのテスト・デバッグを行うサブステップを含むことを特徴とする請求項36記載の並行プログラム作成方法。

【請求項40】 前記第3ステップは、前記複数の並行模擬プログラムの一部の実行結果から不要と判断される並行模擬プログラムを取り除くサブステップを含むことを特徴とする請求項36記載の並行プログラム作成方法。

【請求項41】 前記第4ステップは、前記部分逐次プログラムに対して並行化の候補となるプロセス群を指定するサブステップを含み、

前記第3ステップから前記第4ステップを所定の回数繰り返すステップを更に具備することを特徴とする請求項36記載の並行プログラム作成方法。

【請求項42】 前記第1ステップは、前記第1並行プログラムを逐次化ルールに従って前記逐次プログラムに変換する第1サブステップと、前記逐次プログラムに対して並行性に関する情報を導入するために前記逐次化ルールを修正する第2サブステップと、を含み、

前記第3ステップは、前記テスト・デバッグ情報と、前記並行性に関する情報とに基づいて前記逐次プログラムを並行化して第2並行プログラムに変換するサブステップを含むことを特徴とする請求項23記載の並行プログラム作成方法。

ラム作成方法。

【請求項43】 並行構造を有する第1並行プログラムをテスト実行する第1ステップと、

前記第1ステップによるテスト実行の結果バグのない実行ログを蓄積する第2ステップと、

前記第2ステップにより蓄積された前記実行ログを並行化して第2並行プログラムに変換する第3ステップと、を具備することを特徴とする並行プログラム作成方法。

【請求項44】 前記第1ステップによるテスト実行の結果バグのある実行ログを蓄積する第4ステップと、前記第4ステップにより蓄積された実行ログに基づいて前記第1並行プログラムを修正して、再度実行ログを蓄積する第5ステップと、を更に具備することを特徴とする請求項43記載の並行プログラム作成方法。

【請求項45】 並行プログラムを解析してセクション情報の抽出をおこなうプログラム解析手段と、

前記セクション情報を記憶するセクション情報記憶手段と、

前記セクション情報記憶手段により記憶されたセクション情報に基づいて前記並行プログラムの実行を行なう並行プログラム実行手段と、を具備することを特徴とする並行プログラム実行装置。

【請求項46】 前記セクション情報記憶手段によって記憶されるセクション情報を編集するセクション情報編集手段及び前記セクション情報記憶手段によって記憶されるセクション情報内の各セクションをオブジェクトに変換するコンパイル手段の少なくとも一方を更に具備することを特徴とする請求項45記載の並行プログラム実行装置。

【請求項47】 前記プログラム解析手段が、入力されたプログラムを中間言語に変換する手段と、前記中間言語からセクションを抽出する手段とを更に含むことを特徴とする請求項45記載の並行プログラム実行装置。

【請求項48】 前記実行手段が、実行するセクションの候補を記憶する実行セクション候補記憶手段と、前記実行セクション候補記憶手段によって記憶されるセクションから次に実行するセクションを選択する実行セクション選択手段と、前記実行セクション選択手段によって選択されたセクションを実行するセクション実行手段とを更に含むことを特徴とする請求項45記載の並行プログラム実行装置。

【請求項49】 前記実行セクション選択手段が、実行可能なセクションを一つ選択するモードと実行可能なセクションをすべて選択するモードを含むことを特徴とする請求項48記載の並行プログラム実行装置。

【請求項50】 前記実行セクション実行手段が、前記実行セクション選択手段によって選択されたセクションが複数の時はそれらのセクションを並行に実行することを特徴とする請求項48記載の並行プログラム実行装置。

## 【発明の詳細な説明】

【0001】

【産業上の利用分野】本発明は、並行プログラムの作成方法及びその作成支援装置並びに並行プログラム実行装置に関する。

【0002】

【従来の技術】近年の半導体集積回路技術の進歩により、複雑なプロセッサ及び大容量のメモリが小型かつ低価格で実現できるようになり、多数のプロセッサからなる並行処理システムや分散処理システムが実用化されている。このようなハードウェアに対しては、専用のプログラム、即ち並列プログラムや分散処理プログラム等（以下「並行プログラム」という）を用いなければならない。従って、並行プログラムをいかに効率よく開発するかは、優れたアルゴリズムを検討する場合と同様に重要な課題となっている。

【0003】ところで、プログラム開発においては、プログラム中のバグを見つけ修正すること（即ちテスト・デバッグ）と呼ばれる開発工程がプログラム開発の効率に大きく影響する。しかし、並行プログラムの開発においては、逐次プログラムの開発においては遭遇することのない、並行プログラム特有の問題を考慮する必要がある。この並行プログラム特有の問題とは、並行プログラムを構成する各プロセスは、各プロセス間における相互作用のタイミングにより様々な振る舞いをする可能性があるため、並行プログラム全体が正しく動作しないという問題である。この問題は、並行プログラムの性質に基づく問題であり、一般に「非決定性」と呼ばれる。

【0004】例えば、図116に示す並行プログラムを考慮する。図116(a)において、プロセスP1は、共有メモリMの初期設定(init)を行うプロセス、プロセスP2は、共有メモリMから読み出し(read)を行うプロセス、プロセスP3は、共有メモリMに書き込み(write)を行うプロセスを示す。これらのプロセスをそれぞれ異なるプロセッサで実行するような並列処理システム等で動作させた場合、全部で6通り（図116(b)参照）の動作の組み合わせがあることになる。通常、システムは初期設定で処理を開始するから、今、プロセスP1(init)→P2(read)→P3(write)又はP1(init)→P3(write)→P2(read)の順番でプログラムが動作する場合に正しい結果が得られるものとすれば、残りの4通り（例えばP2→P3→P1）は、初期化が最初に行われないうえ、明らかに正しい結果が得られないことがわかる。

【0005】上記のようにプロセスの振る舞いに関する非決定性は、並行プログラムを動作させる毎に、その時点におけるシステムの状況等によって結果を異なったものになる。従って、この非決定性に関する問題を解決しない限り、その並行プログラムは、テストにおいてたと

え正常に動作することがあっても、常に正常に動作するという保証はない。

【0006】また、非決定性に関するバグは、一般に、逐次プログラムにおけるバグを見つける場合よりも困難である。なぜなら、逐次プログラムにおいては、テスト・デバッグ時にプログラム中の全てのパスを実行することによって動作を確認することができるのに対し、並行プログラムにおいては、全ての組合せ的なパス（即ち、各プロセス中の全てのパスのみならず、プロセス相互間の振る舞い）を考慮してパスを実行しなければならないからである。上記の例のようにプロセスの数が少ない場合にあっては、各プロセス相互間の振る舞いを全て列挙することは比較的容易であるが、実際のプログラム開発では、その数は膨大になり、その組み合わせも膨大なものとなるため、全ての振る舞いを把握することはもはや不可能なものとなる。

【0007】

【発明が解決しようとする課題】上記のように、並行プログラム開発におけるテスト・デバッグは、逐次プログラム開発におけるテスト・デバッグに比較して、非常に困難である。特に、プログラム自体が巨大化した今日においては、このテスト・デバッグは一層困難になっている。

【0008】更に、並行プログラム特有の問題である非決定性によって再現性が保証されないことからテスト・デバッグが困難となり、また予期せぬエラーが出現するという問題を有する。

【0009】本発明は、並行プログラムのテスト・デバッグを容易に実現でき、並行プログラムの開発を効率よく行うことができる並行プログラムの作成方法及びその作成支援装置を提供することを目的とする。

【0010】本発明の他の目的は、並行プログラムを逐次的に実行することにより再現性を保証し、効果的なテスト・デバッグを行なうことを可能とすると共に、テスト・デバッグ後に一部分を並行化して実行することにより、非決定性による予期せぬエラーを排除した安全な並行プログラムの実行が可能な並行プログラム実行装置を提供することである。

【0011】本発明は並行プログラムを一旦逐次化し、逐次化したプログラムに対してテスト・デバッグを行い、テスト・デバッグが完了した時点で、プログラムの並行性を復元することを骨子とする。

【0012】

【課題を解決するための手段】本発明は、上記の課題を解決するために次のような手段を講じた。並行プログラムのプログラミングの困難さは、「人間の思考は本来逐次的であり、並行に動くものをありのままでは論理的に認識することは困難である」ことに起因する。そこで、本発明は並行プログラムをいったん逐次化し、逐次化したプログラムに対してプログラミング、テスト・デバ

グを行う。これは、従来の逐次プログラミングと同じレベルの困難さである。そして、テスト・デバッグが完了した時点で並行性を、そのテスト・デバッグ情報を用いて、自動的に復元する。

【0013】上記のようなプログラミングのスタイルを「超逐次プログラミング」と呼ぶ。この「超逐次プログラミング」によれば、従来の手法におけるプログラミングの困難さを解決できる。本発明の基本コンセプトは、以下の3つのステップ（或いは手段）から構成される。

【0014】（1） 並行プログラムを逐次化して超逐次プログラムを生成するステップ（手段）。

【0015】（2） 超逐次プログラムに対して、作業（プログラミング、テスト・デバッグ、並行性の導入）を行うステップ（手段）。

【0016】（3） 作業が完了した超逐次プログラムを並行化して並行プログラムを作成するステップ（手段）。

【0017】ここで、「超逐次プログラム」とは、オリジナルの並行プログラムの並行構造に関する情報を保ちながら逐次化したプログラムをいう。

【0018】本発明に係る並行プログラム作成装置は、並行構造を有する第1並行プログラムを逐次実行可能な逐次プログラムに変換する逐次化手段と、前記逐次プログラムのテスト・デバッグを行い、テスト・デバッグ情報を作成するテスト・デバッグ手段と、テスト・デバッグ後の前記逐次プログラムを前記テスト・デバッグ情報に基づいて並行化することにより第2並行プログラムに変換する並行化手段とを具備することを特徴とする。また、本発明に係る並行プログラムの作成方法は、並行構造を有する第1並行プログラムを逐次実行可能な逐次プログラムに変換する第1ステップと、前記逐次プログラムのテスト・デバッグを行い、テスト・デバッグ情報を作成する第2ステップと、テスト・デバッグ後の前記逐次プログラムを前記テスト・デバッグ情報に基づいて並行化することにより第2並行プログラムに変換する第3ステップとを具備することを特徴とする。

【0019】本発明の好ましい態様を以下に列挙する。

（1） 前記逐次プログラムに対して並行性に関する情報を導入すること。この並行性に関する情報は、例えば、後述する良い非決定性に関する情報を含む。

（2） 第1並行プログラムの並行構造を解析し、この並行構造と逐次プログラムをテストして得られた実行結果を用いて逐次プログラムの並行化を行うこと。

【0020】（3） 第1並行プログラムの逐次プログラムに変換されるセクションの並行構造と、逐次プログラムのセクションの逐次構造をそれぞれ解析し、第1並行プログラムの並行構造に関する相互関係及び逐次プログラムの逐次構造に関する相互関係をグラフ情報として表示する。このグラフ情報の表示においては、所定のセクション群をノードとし、第1並行プログラムの並行構

造に関する相互関係を第1アークとし、逐次プログラムの逐次構造に関する相互関係を第2アークとしてグラフ情報を表示する。そして、逐次プログラムのうち選択されたセクションを並行化して部分的に逐次構造を有する部分逐次プログラムに変換し、この部分逐次プログラムを並行化して第2並行プログラムに変換する。

【0021】ここで、第1並行プログラムを逐次プログラムに変換するステップにおいて、逐次プログラムの所望の実行結果が得られるまで逐次プログラムのテスト・デバッグを行うステップを含ませるか、又は逐次プログラムを部分逐次プログラムに変換するステップにおいて、所望の実行結果が得られるまで部分逐次プログラムのテスト・デバッグを行うステップを含ませるか、或いはこれら両方のテスト・デバッグステップを含ませてもよい。

【0022】更に、逐次プログラムを部分逐次プログラムに変換するステップに、部分逐次プログラムの逐次構造を解析するステップを含ませた上で、グラフ情報の表示と並行化セクションの選択及び部分逐次プログラムへの変換のステップを所定回数繰り返すようにしてもよい。

【0023】（4） 第1並行プログラムを逐次プログラムに変換する逐次化に際して、この第1並行プログラムを実行してそれによる実行ログを保存するとともに、この保存された実行ログ及び第1並行プログラムを解析し、この解析結果に基づいて、保存された実行ログを並べ替える。実行ログ及び第1並行プログラムの解析においては、例えば保存された実行ログ及び第1並行プログラムからプロセスの先行関係を抽出し、これを先行関係情報として保存する。

【0024】（5） 逐次プログラムに対する並行性に関する情報の導入に際して、逐次プログラムのプロセスの流れを制約と遷移条件からなるフィールドに変換し、このフィールドをチューニングすることによって行う。更に、このフィールドを表すフィールドデータを表示する。

【0025】（6） 逐次プログラムに対して並行化の候補となるプロセス群を指定し、このプロセス群の実行順序を入れ換えて逐次プログラムを複数の並行模擬プログラムに変換した後、これら複数の並行模擬プログラムを部分的に逐次構造を有する1つの部分逐次プログラムに変換し、この部分逐次プログラムを並行化して第2並行プログラムに変換する。

【0026】ここで、第1並行プログラムを逐次プログラムに変換するステップにおいて、逐次プログラムの所望の実行結果が得られるまで該逐次プログラムのテスト・デバッグを行うステップを含ませるか、又は複数の並行模擬プログラム群を1つの部分逐次プログラムに変換するステップにおいて、並行模擬プログラム群の所望の実行結果が得られるまで複数の並行模擬プログラムのテ

スト・デバッグを行うか、或いはこれら両方のテスト・デバッグステップを含ませてもよい。また、逐次プログラムに対して並行化の候補となるプロセス群を指定する際、第1並行プログラムを解析し、この解析結果から並行化の候補となるプロセス群を抽出してもよい。また、逐次プログラムを複数の並行模擬プログラムに変換するステップにおいて、複数の並行模擬プログラムの一部の実行結果から不要と判断される並行模擬プログラムを取り除くステップを含ませてもよい。更に、複数の並行模擬プログラムを1つの部分逐次プログラムに変換するステップにおいて、部分逐次プログラムに対して並行化の候補となるプロセス群を指定するステップを含ませ、逐次プログラムを複数の並行模擬プログラムに変換するステップと、複数の並行模擬プログラムを1つの部分逐次プログラムに変換するステップを所定の回数繰り返してもよい。

【0027】(7) 第1並行プログラムの逐次プログラムへの変換を所定の逐次化ルールに従って行う場合、この逐次化ルールを修正することによって、逐次プログラムに対して並行性に関する情報を導入する。

【0028】(8) プロセス群がメッセージ情報を交換しながら並行して動作する実行環境に用いられる並行プログラムの作成を支援する並行プログラム作成支援装置において、並行プログラムのプロセス群の実行履歴であるログ情報を逐次化ルールとして取得して記憶し、この記憶したログ情報を修正可能とする。記憶されているログ情報に基づいてプロセス群を逐次的に起動制御し、また記憶されたログ情報を並行化して第2並行プログラムに変換する。

【0029】ここで、ログ情報修正手段は、前記ログ情報記憶手段に記憶されているログ情報を時系列に表示する表示手段と、前記表示手段により時系列に表示されたログ情報内のデータの順序の入れ換えを指示するための入れ替え指示手段と、前記入れ替え指示手段による指示に従って前記ログ情報記憶手段に記憶されているログ情報を書き換える書換手段とを含むことを特徴とする。また、ログ情報修正手段は、プロセス間の処理タイミングの非決定性を導入する非決定性導入手段を含むことを特徴とする。

【0030】(9) プロセス群がメッセージ情報を交換しながら並行して動作できる実行環境で前記プロセス群が実行順序規定情報に従って動作するシステムにおいて、実行順序規定情報を分割し、この分割された実行順序情報に基づいてプロセス群を起動制御する。

【0031】この場合、実行順序規定情報を分割するための基準を与える分割基準指定手段を保持する手段を更に具備してもよい。また、メッセージ交換の履歴を実行順序規定情報として用いるとともに、分割基準指定手段における分割基準としてメッセージ中の宛先プロセス情報を基準として用いる。また、メッセージ交換の履歴を

実行順序規定情報として用い、分割基準指定手段における分割基準として前記メッセージ中の宛先プロセス情報を基準として用いるとともに、プロセス制御手段を各プロセス毎に保持する手段を具備する。更に、メッセージ交換の履歴を実行順序規定情報として用い、分割基準指定手段における分割基準として前記メッセージ中の宛先プロセス情報を基準として用いるとともに、プロセス制御手段を各プロセス毎に保持する手段と、実行順序情報分割手段によって分割された実行順序情報をそれぞれ別々に保存する分割実行順序情報保存手段を具備し、プロセス制御手段は該プロセスに対応する分割実行順序情報保存手段に保存されている分割実行順序情報に基づいてプロセスを起動制御する。また、プロセス群の実行履歴情報であるログ情報を実行順序規定情報として用いてもよい。

【0032】(10) 第1並行プログラムをテスト実行して、そのテスト実行の結果バグのない実行ログを蓄積し、この蓄積されたバグのない実行ログのみを並行化して第2並行プログラムに変換する。更に、テスト実行の結果バグのある実行ログを蓄積し、この蓄積されたバグのある実行ログに基づいて第1並行プログラムを修正する。

【0033】また、本発明の並行プログラム実行装置は、並行プログラムを解析してセクションを抽出し、それらのセクション間の実行時の同期を制御するための実行順序に関するルールを抽出するプログラム解析手段と、実行時のセクション間の同期を調節するために実行順序に関するルールを編集し、主に実行の効率化のために、抽出された当該セクションを融合・分割する編集手段と、セクションをオブジェクトに変換するコンパイル手段と、該実行順序に関するルールに従いセクション単位で逐次又は並行に実行を行なう実行手段を具備する。

【0034】

【作用】上記手段を講じた結果、次のような作用が生じる。本発明では並行プログラムを一旦逐次化し、逐次化したプログラムに対してテスト・デバッグを行うことにより、従来の並行プログラムのプログラミングより遥かに容易な逐次プログラミングと同じレベルの困難さで並行プログラムのテスト・デバッグが可能となる。

【0035】また、本発明では、いったん逐次化されたプログラムに対して意図的に並行性に関する情報(良い非決定性)を導入できるので、意図しない並行性(悪い非決定性)に基づいて発生するバグを回避でき、高信頼化が達成できる。

【0036】上記のグラフ情報によって並行化及び逐次化の情報をユーザに対して同時に提示することにより、ユーザは第1並行プログラムの並行構造を考慮しつつ、良い非決定性部分の指定をすることができるようになる。また、並行プログラム記述レベルにおける良い非決定性部分の指定・解除ではなく、グラフ情報に対して良



い非決定性部分を指定・解除を行うことのできるため、高度な並行プログラミング技術を必要とすることなく、容易に並行プログラムの開発をすることができるようになる。

【0037】第1並行プログラムを逐次プログラムに変換する際、第1並行プログラム及びその実行ログを解析し、この解析結果に基づいて実行ログを並べ替えるようにすれば、並べ替え後の実行ログを表示してユーザに提示することによって、並行プログラムの実行過程の理解が容易となり、テスト・デバッグの効率が向上する。逐次プログラムに並行性に関する情報を導入する際、逐次プログラムのプロセスの流れを制約と遷移条件からなるフィールドに変換し、更にそのフィールドデータを表示することによって、フィールドを対話的・視覚的に編集することで並行性に関する情報を効果的に導入し、バグのない並行プログラムが効率的に作成される。

【0038】第1並行プログラムから逐次化された逐次プログラムに対して並行化の候補となるプロセス群を指定し、このプロセス群の実行順序を入れ換えて逐次プログラムを複数の並行模擬プログラムに変換した後、これら複数の並行模擬プログラムを部分的に逐次構造を有する1つの部分逐次プログラムに変換し、この部分逐次プログラムを並行化して第2並行プログラムに変換することにより、部分逐次プログラム上で並行プログラムの動作を十分に確認することができる。また、部分逐次プログラムに対して並行化の候補となるプロセス群を指定することで、逐次構造プログラムを段階的に並行プログラムへ変換することができる。更に、並行模擬動作系列に基づく逐次実行で正しく動作することが確認された非決定性のみを許容する並行性に関する情報を導入することで、正しく動作する並行プログラムを得ることができる。これらによって、並行プログラムのテスト・デバッグが容易となる。

【0039】プロセス群がメッセージ情報を交換しながら並行して動作する実行環境に用いられる並行プログラムの作成を支援する並行プログラム作成支援装置において、第1並行プログラムのプロセス群の実行履歴であるログ情報を逐次化ルールとして取得して記憶し、このログ情報を修正可能とするとともに、記憶されているログ情報に基づいてこれに基づいてプロセス群を逐次的に起動制御し、記憶されたログ情報を並行化して第2並行プログラムに変換することにより、ソースプログラムとしての並行プログラムを修正することなく、ログ情報の修正で処理タイミングの非決定性に起因する不具合を解決できる。これにより、処理タイミングの非決定性が内在する並行／並列／分散プログラムの開発が容易となる。また、ユーザの意図する良い非決定性のみを容易に導入することができるため、並行プログラムとしての柔軟性、再利用性及び拡張性を維持することもできる。

【0040】プロセス群がメッセージ情報を交換しながら

ら並行して動作できる実行環境でプロセス群が実行順序規定情報に従って動作するシステムにおいて、実行順序規定情報を分割し、つまり並行プログラムを逐次化して得られた全プロセスの集中ログ情報を各プロセス毎に分割し、この分割された実行順序情報に基づいてプロセス群を起動制御することにより、無害の非決定性を自然に導入し、集中ログ情報に基づく逐次プログラムの実行時と同一結果を高い処理効率で得ることが可能となる。

【0041】第1並行プログラムをテスト実行してそのテスト実行の結果、その1つであるバグのない実行ログを蓄積し、この蓄積されたバグのない実行ログのみを並行化して第2並行プログラムに変換することにより、テストで通過したタイミングだけを許容するようにプログラムが動くようになるため、テストしなかったことで残存したバグに陥ることを回避でき、信頼性が向上する。

【0042】本発明の並行プログラム実行装置によれば並行プログラムの逐次又は部分並行実行を行なうことが可能となるので、従来の逐次プログラムと同様に再現性を保証することができ、効果的かつ安定したテスト・デバッグを行なうことができる。また、逐次実行から部分並行実行に容易に切替えることができるため、逐次化してテスト・デバッグをおこなったプログラムを非決定性に影響されない安全な並行実行を可能とする。他に、セクションの実行順序に関するルールを編集して実行時のセクション間の同期を制御することにより、効果的なテスト・デバッグを行なえとともにプログラムの効率化の指針とすることもできる。

【0043】更に、本発明の並行プログラム実行装置は、本発明の並行プログラム作成支援装置或いは作成方法と組み合わせて用いることもできる。この場合には、並行プログラム実行装置は、第1並行プログラム（ソースプログラム）の実行手段として、機能する。更に、並行プログラム実行装置は、本発明の並行プログラム作成支援装置或いは作成方法により作成された並行プログラムの検証用としても利用できる。

【0044】

【実施例】以下、図面を参照して本発明の実施例を説明する。本発明の実施例の説明を行う前に、本発明において使用する用語を以下のように定義する。

【0045】並行システム：論理的な並行性を持つシステム。並行プログラムが動くシステムは並行システムである。並列計算機や分散処理システムは並行システムである。

【0046】並行プログラム（CP）：論理的に並行に動くモデルに基づいて記述されたプログラム。複数のCPUから構成される並列計算機上で、論理的にも物理的にも並行に動くプログラム（並列プログラム）は、並行プログラムに含まれる。また、単一のCPU上で物理的には逐次的に動く場合でも、マルチタスクシステムのように論理的に並行であれば、並行プログラムに含まれ

る。

【0047】逐次プログラム：論理的に逐次に動くモデルに基づいて記述されたプログラム。

【0048】超逐次プログラム（HSP）：並行プログラムにメタレベル（実行管理レベル）の制御を付加することによって、逐次的に動くようにしたプログラム。例えば、並行プログラムとして記述されたプログラムを、実行管理レベルのスケジューラを付加することによって逐次的に動くようにしたプログラムは、超逐次プログラムである。超逐次プログラムは後述する非決定性を持たないので、外部からの入力と同じならばその挙動は必ず再現性を持つ。また、超逐次プログラムにおいて、部分的に並行性（非決定性）を導入することもできる。部分的に並行性（非決定性）が導入された超逐次プログラムを、特に、部分超逐次プログラム（PHSP）、並行性（非決定性）が全くなく完全に逐次化されたプログラムを完全超逐次プログラムと呼ぶこともある。

【0049】非決定性：入力が同じにもかかわらず、処理のタイミングによってシステムの挙動が変わること。並行プログラムの実行において、非決定性は本質的側面である。ある意味で、非決定性のないプログラムは論理的に逐次プログラムと等価である。

【0050】良い非決定性：ユーザの意図する非決定性をいう、即ち、ユーザの仕様、その他実現要求に含まれる非決定性。この非決定性により、外部（環境）からの非決定的な入力に適切に応答できるようになる。

【0051】悪い非決定性：ユーザが予想しなかった非決定性。ユーザの思考回路は逐次的であるため、設計時には意図しなかったケースが実際の並行プログラムの実行では多々発生する。

【0052】無害な非決定性：その非決定的な選択肢の選択が、最終的な結果に影響しないもの。「超逐次プログラミング」の並行化装置において、並行化する対象は、この「無害な非決定性」を含む。

【0053】デフォルト逐次化：実行管理レベルで逐次化制御を行なうこと。

【0054】並行模擬プログラム：超逐次プログラムにおいて、並行化の候補となる特定の範囲について生成された並行模擬動作系列の集合。

【0055】・並行模擬動作系列：超逐次プログラムにおいて並行化の候補となる特定の範囲について並行プログラムの動作を逐次プログラム上で模擬することを目的として、逐次実行の順序を任意又は意図的に入れ替えて生成される1つの動作列のこと。一般に、1つの並行化範囲には複数の並行模擬動作系列が生成されるので、その全体を1つの並行模擬プログラムという。超逐次プログラム全体では複数の並行模擬プログラムが生成される。

【0056】図1は、本発明に係る並行プログラム作成支援装置を実現するためのコンピュータシステムの構成

例を示す図である。図1において、N台のプロセッサ1-1、1-2、…、1-Nは、並行プログラムを同時に実行することができ、I/Oインタフェース2を介して共有メモリ3及び周辺装置とアクセスすることができる。周辺装置は、入力装置4と出力装置5及び外部記憶装置6から構成される。入力装置4は、キーボードとポインティングデバイス等とからなり、各種コマンドやデータのを入力をするために用いられる。出力装置5は、CRTディスプレイ等とからなり、ソースプログラムやテスト・デバッグ状況に関する情報等をテキスト又はグラフィック表示することにより、ユーザに提示する。ユーザは、これら入力装置4及び出力装置5を用いて、対話的にコンピュータを操作することができる。

【0057】外部記憶装置6は、磁気ディスクや光磁気ディスク等とからなり、ソースプログラムやテスト・デバッグ状況に関する情報を書き込み又は読み出すことができるようになっている。

【0058】なお、上記説明したコンピュータシステムの構成は、これにこだわる必要はなく、例えば、複数の計算機をネットワークを用いて接続した、いわゆる分散ネットワークを用いて構成してもよい。

【0059】上記のように構成されたコンピュータシステムにおいて、本発明における並行プログラムの作成は、以下のようにして実現される。

【0060】（実施例1）実施例1では、テスト・デバッグは部分超逐次プログラムに対して行い、好ましい実施例として良い非決定性の導入を超逐次プログラムに対して行う。なお、以下の実施例において、良い非決定性を導入した例を説明するが、必ずしも良い非決定性を導入しなくても良い。

【0061】図2は、実施例1に係る並行プログラム作成支援装置の概略構成を示す図である。実施例1に係る並行プログラム作成支援装置は、逐次化装置12と、テスト実行装置15と、デバッグ装置16と、非決定性導入装置17と、並行化装置18とを具備する。

【0062】逐次化装置12は、第1CPファイル記憶部11に記憶されたソースプログラム（以下、「第1並行プログラム」と称する）を逐次化ルール記憶部13に記憶された逐次化ルールに基づいて超逐次プログラムに変換して、その結果がHSPファイル記憶部14に記憶される。第1CPファイル記憶部11には、モデル化され並行プログラミング言語で記述された第1並行プログラムが格納されている。この第1並行プログラムには、バグが存在する可能性がある。第1並行プログラムを記述する並行プログラミング言語には、例えば以下のものがある。

- (a) Concurrent PASCAL
- (b) ADA
- (c) GHC
- (d) Modula 3

(e) Occam

(f) c++C

テスト実行装置15及びデバッグ装置16は、それぞれ、HSPファイル記憶部14に記憶された超逐次プログラムのテスト・デバッグを行う。

【0063】非決定性導入装置17は、第1並行プログラムを超逐次プログラムに変換する際に、良い非決定性を導入する。

【0064】並行化装置18は、HSPファイル記憶部14に記憶された超逐次プログラムのテスト・デバッグ情報に基づいて、超逐次プログラムを並行化して、第2並行プログラムを生成する。この第2並行プログラムは、第2CPファイル記憶部19に記憶される。

【0065】図3は、実施例1に係る並行プログラム作成方法の概略手順を示すフローチャートである。

【0066】(1) ステップA1：モデル化  
対象の並行システムに対し、並行性を用いた自然なモデル化を行う。また、並行システムの各プロセス構造を決定する。更に、該各プロセス内を並行プログラム等を用いたプログラミングにより、並行構造を有する並行プログラムをソースプログラムとして記述する。なお、「並行構造を有する」としたのは、一般に並行プログラムは、その全てが並行構造で構成されているわけではなく、逐次性を用いたモデル化の方がより自然である場合には、その部分は逐次構造である場合があるからである。なお、このソースプログラムには、バグが潜在的に存在する可能性がある。

【0067】(2) ステップA2：逐次化  
デフォルト逐次化によって、第1並行プログラムを逐次構造の超逐次プログラムに変換する。本実施例では、メタレベルにおいて逐次性を導入する。ここで、メタレベルとはソースプログラム（すなわち、第1並行プログラム）そのもののレベルではなく、ソースプログラムの実行を管理するレベルをいう。例えば、並行プログラムで記述されたソースプログラムを、それとは別に管理されるスケジューラによって逐次的に実行することを保障したプログラムのソースプログラムに変換する。

【0068】(3) ステップA3：超逐次プログラムのテスト・デバッグ

超逐次プログラムのテスト・デバッグを行う。テスト実行装置により超逐次プログラムをテスト実行した結果に基づいて、デバッグ装置により超逐次プログラムからバグを除去する。ここでのテスト・デバッグは、逐次プログラムにおける通常のテスト・デバッグ方法と同様に行うことができる。超逐次プログラムが正常に動作することが保障されるまで、テスト・デバッグを行う。

【0069】(4) ステップA4：超逐次プログラムに対する良い非決定性の導入

テスト・デバッグが行われた超逐次プログラムに対して、良い非決定性に関する情報（並行性に関する情報）

を与える。これにより超逐次プログラムは、非決定性に関する情報を一部に持つことで、部分超逐次プログラムとなる。良い非決定性に関する情報の導入方法は後述する。

【0070】(5) ステップA5：部分超逐次プログラムのテスト・デバッグ

ステップA4で得られた部分超逐次プログラムに対して、テスト・デバッグを行う。すなわち、ステップA4で良い非決定性に関する情報が導入された超逐次プログラムをテスト・デバッグする。

【0071】(6) ステップA6：部分超逐次プログラムに対する良い非決定性の導入・拡大

ステップA5でテスト・デバッグが行われた部分超逐次プログラムに対して、良い非決定性に関する情報を追加する。ステップA5～ステップA6を所定の回数繰り返し、良い非決定性を徐々に拡大していく。

【0072】(7) ステップA7：並行化コンパイル  
良い非決定性に関する情報が導入された部分超逐次プログラムのうち、無害な非決定性部分を抽出し、その部分を並行化することで、部分超逐次プログラム全体を並行プログラム（すなわち、第2並行プログラム）に還元する。すなわち、並行化に関する情報が導入されなかった部分に関しては、デフォルト逐次化で与えられた逐次性を並行プログラム中に反映させて（例えば、ソースプログラム自体に埋め込む）、メタレベルのデフォルト逐次性は解除する。

【0073】図4～図7を参照して、本発明をより具体的に説明する。

【0074】図4は、簡単な並行プログラムの一例を示す図である。

【0075】図4の並行プログラムは、プロセスP1とプロセスP2とから構成されている。これらのプロセスは、並行プログラムのソースコードがコンパイルされることにより生成された実行モジュールがコンピュータ上で実行された時に初めて実体化するものであり、プロセスP1とプロセスP2とにそれぞれ対応する並行プログラムが必ずしも物理的に別の記憶媒体に記憶されている必要はない。また、メモリMはここでは共有メモリ(shared memory)を表し、並行プログラムのアクセス命令によって書き込み/読み込み等のアクセスを行うことができる。図4において実線矢印はプロセスP1、P2が実行された時に共有メモリMとの間でアクセスが行われることを示す。

【0076】CPファイル記憶部11に格納された第1並行プログラムは、ユーザによる入力装置4からの指示により引き出され、逐次化装置12に入力される。逐次化装置12に入力された第1並行プログラムは、逐次化ルール記憶部13に格納された逐次化ルールに従って、メタレベルにおいてデフォルト逐次性が導入されることにより、超逐次プログラム(HSP)に変換され、HS

Pファイル記憶部14に記録される。

【0077】逐次化ルールの代表的なものには以下のものがある。

- (a) プロセスに優先度を導入するルール。
- (b) プロセス内の処理単位（オブジェクト指向におけるメソッド）に優先度を導入するルール。
- (c) 具体的な実行ログに基づく逐次化ルール。
- (d) メッセージの到着先の実行を優先する逐次化ルール。
- (e) メッセージの送信元の実行を優先する逐次化ルール。

【0078】図4の下方に記載した“P1>>P2”が図4の並行プログラムに対する逐次化ルールの一例であり、二つのプロセスP1、P2に対してP1をP2より優先的に動作させるという優先度を与えることを表している。これは上記(a)のルールに相当する。この逐次化ルールは、例えば第1並行プログラムの先頭で宣言しておき、プログラム本体とともにコンパイルすることにより並行プログラムに導入するようにしても良いし、並行プログラムとは別にファイルに記述しておき、並行プログラムが実行時にオペレーティングシステムやスケジューラが解釈することにより導入するようにしても良い。なお、本実施例では逐次化ルールを“>>”で示したが、これに限らず任意の記号を用いることができる。

【0079】図5は、メタレベルにおいてデフォルト逐次性が導入された超逐次プログラムHSPの概念図であり、プロセスP1とプロセスP2とはスケジューラSによって管理されていることを示しており、図5において、破線矢印は図4に示した逐次化ルール（“P1>>P2”）に従って、スケジューラSがプロセスP1を実行後、プロセスP2を実行することを示す。なお、この超逐次プログラムHSPの概念図は、図5の下方に記載した式、

HSP=P1|P2|S

のように記述するものとする。これは、超逐次プログラムHSPはプロセスP1とプロセスP2とスケジューラSとから構成されていることを示す。ここで、逐次化ルールはスケジューラのスケジューリング規則に対応する。

【0080】ユーザは、この超逐次プログラムHSPを出力装置5によって見ることができる。この超逐次プログラムHSPは、ユーザによる入力装置4からの指示によってテスト実行装置15に入力され、テスト実行が行われる。テスト実行装置15は、テスト実行の結果（実行ログ）を出力装置5に提示する。ユーザは、このテスト実行の結果に基づいて、入力装置4をデバッグ装置16として用いて、超逐次プログラムHSPに対し所定のテスト・デバッグを行うことができる。具体的なテスト・デバッグ技術には、

- (a) ソースコードのトレーサ

- (b) ブレークポイント

- (c) アニメーション

等がある。

【0081】図6は、この時のテスト・デバッグのイメージを表す図である。図6において、出力装置5上のデバッグ画面60には、各種ウィンドウ61～65がオープンされ、各種情報が表示されており、これらウィンドウ61～65は、適宜オープン・クローズすることが可能である。なお、ここでのテスト・デバッグは、基本的には公知のデバッグ装置を用いることができ、具体的には、UNIXワークステーション上のdbx tool等が知られている。

【0082】ユーザは、超逐次プログラムHSPに対して所定のテスト・デバッグを行った後、入力装置4から再度テスト実行の指示を与える。これにより、テスト・デバッグの行われた超逐次プログラムHSPは、テスト実行装置15に入力され、再度テスト実行が行われる。このテスト・デバッグは、超逐次プログラムHSPが正常に動作することを確認するまで繰り返し行われる。テスト・デバッグにより正常に動作することを確認した時点で、非決定性導入装置17により超逐次プログラムに対して良い非決定性に関する情報を部分的に導入していく。非決定性導入装置17より導入された良い非決定性に関する情報は、超逐次プログラムHSPに反映され、HSPファイル記憶部14に記録される。なお、非決定性導入装置17については後述する。

【0083】図7は、超逐次プログラムに良い非決定性に関する情報が導入された状態の一例を示す図である。ここで、並行プログラムの各プロセスの実行単位を「セクション」と呼ぶと、図7はセクションS1とセクションS2とに分けられたプロセスP1と、セクションS3とセクションS4とに分けられたプロセスP2について、セクションS2とセクションS3の優先度を同じものとするにより、非決定性が導入された状態を示す。

【0084】より具体的には、実行単位毎に異なる優先度を与えている逐次化ルールに対し所定の部分に良い非決定性に関する情報を与える（例えば、優先度を同じにしたいプロセスをマウスで指定する）ことにより、当該部分については優先度を同じとして、並行に実行可能にする。図7の例では、S1～S4の4つのセクションのうち、「S2である“write1”とS3である“read2”の優先度を同じにする」という情報（S2=S3）を良い非決定性として導入している。すなわち、優先度が同じプロセスは、どちらが先に実行されてもかまわないので、非決定性を持つ。

【0085】次に、非決定性導入装置17により良い非決定性に関する情報が導入された超逐次プログラム（部分超逐次プログラムPHSP）は、ユーザからの指示に従ってテスト実行装置15によりテスト実行が行われ、

デバッグ装置 16 によりテスト・デバッグが行われる。この場合、部分超逐次プログラム PHSP の振る舞いは、良い非決定性に関する情報の導入された部分については非決定的な振る舞いをするので、その振る舞い全てについてテスト・デバッグを行うことが好ましい。このようにして、テスト・デバッグ及び良い非決定性に関する情報の導入を繰り返し、良い非決定性に関する情報を徐々に付加していく。

【0086】良い非決定性に関する情報がインクリメンタル(incremental)に導入されて得られた部分超逐次プログラム PHSP は、ユーザからの指示により並行化装置 18 に入力される。並行化装置 18 は、部分超逐次プログラム PHSP のうち無害な非決定性部分を抽出し、部分超逐次プログラム PHSP 全体を並行化する。即ち、並行化装置 18 は、導入された良い非決定性と無害な非決定性に関してデフォルト逐次性を解除し、並行プログラム CP (第 2 並行プログラム)としてファイルに記録する。ここで、良い非決定性と無害な非決定性以外はデフォルト逐次化で与えられた逐次性が並行プログラム CP に反映されなければならない。ユーザは、この第 2 並行プログラムを出力装置 5 によって見ることができるとともに、最終的なテスト・デバッグを行うことができる。

【0087】実施例 1 は、本発明の基本的な実施例を示したが、以下、更に詳細な実施例を実施例を説明する。但し、以下の実施例においては、実施例 1 と共通の部分や相対応する部分については同一符号を付して説明を簡略化するか又は省略し、相違点を中心に説明する。

【0088】(実施例 2) 実施例 2 では、実施例 1 と同様に、次のような並行プログラムを対象とする。並行プログラムは複数のプロセスから構成される。並行プログラムは、共有メモリ型のマルチプロセッサで実行される。各プロセス毎にプロセッサ(CPU)が割り当てられる。各プロセスの同期は、同期基本命令と共有メモリ型で実現される。

【0089】上記の並行プログラムに本発明を適用した実施例を説明する。図 8 は、実施例 2 に係る並行プログラム作成支援装置の概略構成を示すブロック図であり、図 9 は、実施例 2 に係る並行プログラム作成方法の概略手順を示すフローチャートである。図 8 が、図 2 と異なる点は、セクション設定装置 7 を更に具備し、テスト実行装置 15 とデバッグ装置 16 とを、具体的に、テスト実行装置 15、修正装置 9、及び解析装置 10 とし、解析装置 10 で解析された解析情報を記憶する解析情報記憶部 20 を有する点である。解析装置 10 では、超逐次プログラムを解析し、解析情報として後述する先行制約を抽出する。他の構成部分は実施例 1 と同様であるので、説明を省略する。なお、図 8 には、並行化装置 18 で、超逐次プログラムの並行化を行う際に参照される並行化ルールを記憶する並行化ルール記憶部 21 を入れて

いる。

【0090】セクション設定装置 7 は、第 1 並行プログラムの各プロセスをいくつかのセクション(プログラム単位)に分割する。

【0091】修正装置 9 は、テスト実行装置 15 によるテストの結果バグがあれば、修正を行う。

【0092】テスト実行装置 15 は、実施例 1 と同様に、超逐次プログラムのテストを行う。

【0093】解析情報記憶部 20 は、解析装置 10 で解析された情報を記憶する。

【0094】実施例 2 の動作を図 9 のフローチャートを参照して説明する。なお、図 9 において、図 3 のフローチャートと同一動作には同一の符号を付す。

【0095】(1) ステップ A1: モデル化  
対象の並行システムに対し、並行性を用いた自然なモデル化を行う。また、並行システムの各プロセス構造を決定する。更に、該各プロセス内を並行プログラム等を用いたプログラミングにより、並行構造を有する並行プログラムをソースプログラムとして記述する。このソースプログラムには、バグが潜在的に存在する可能性がある。

【0096】(2) ステップ B1: セクションの設定  
セクション設定装置 7 により、設計者が第 1 並行プログラムの各プロセスをいくつかのセクション(単位)に分割する。ここで、第 1 並行プログラム中の同期命令は自動的に単独のセクションとする。ここで、セクションはプロセスの処理の単位であり、以下のステップでは、セクションを単位として、逐次化及び並行化を行う。この場合において、設計者によるセクション設定がなくても構わない。この場合は、同期命令で区切られる区間が自動的にセクションになる。

【0097】セクションの設定は、プロセスのソースコードを分割し、分割された各区間にセクション ID を設定することで実現できる。ソースコード分割の一例としては、図 10 に示すように、区切りポイントを挿入し、区切りポイントから次の区切りポイントまでの処理をセクションとする方法がある。ここで、上記のように同期命令の前後には自動的に区切りポイントが挿入される。

【0098】(3) ステップ A2: 逐次化  
逐次化ルールに基づき、逐次化装置により、第 1 並行プログラムを逐次化する。逐次化ルールの一例としては、プロセスに優先度を導入する方式がある。この優先度に基づいて実行すれば、実行時の非決定性は存在しないので、超逐次プログラムとみなすことができる。このようにして並行構造に関するプログラム情報を持ちながら逐次化されたプログラムを超逐次プログラムとする。逐次化方式の一例としては、「プロセスの優先度」に基づく方式がある。この方式によれば、プロセスに予め固定の優先度を設定し、優先度が高いプロセスのセクションの実行を優先することにより、非決定性のない逐次的な実

行順序が得られる。別の逐次化方式の例としては、図1に示すような「同期命令のwait側の実行を優先する方式（逐次化ルール1）」や「同期命令のsend側の実行を優先する方式（逐次化ルール2）」等がある。ここで、超逐次プログラムは、3種のプログラム情報、すなわちセクション情報と、プログラム構造情報と、逐次化情報とから構成される。セクション情報は、セクションの識別子（ID）とセクションの属するソースコードの情報である。プログラム構造情報は、オリジナルの並行プログラムにおける、プロセス毎のセクションの実行順序情報、更に、異なるプロセスのセクション間のデータ依存関係の情報である。逐次化情報は、逐次化によるグローバルなセクションの実行順序情報であり、同時によい並行性のに関する情報も持つ。一例としては、並行システムのモデル化手法であるベトリネットで逐次化情報を表現する方法がある。

【0099】（4） ステップA3：超逐次プログラムに対するテスト・デバッグ

超逐次プログラムに対して、テスト実行装置15でテストを行い、バグがある場合は、修正装置9でデバッグ／修正を行う。また、プログラムの修正が同期命令等の並行構造の変更にも及ぶ場合は、ステップA1に戻り、プログラム作成装置8でモデル化を行い、再度セクションの設定及び逐次化を行う。ここで、プログラム6は逐次化されているので、テスト・デバッグは逐次プログラムなみに容易になる。

【0100】（5） ステップA4：超逐次プログラムに対する良い非決定性の導入

非決定性導入装置17により、出力装置5に超逐次プログラムの構造が示される。設計者はこの構造を見ながら非決定性導入装置17により良い非決定性による並行性を明示的に導入できる。この時、解析装置10により抽出された超逐次プログラムのプログラム情報により、設計者による良い非決定性の導入を支援する。良い非決定性の導入の必要がなければ、ステップB2に進む。良い非決定性の導入方式として、例えば、超逐次プログラムの逐次化情報がベトリネットで表現されている場合には、プログラム構造を保存する範囲でのベトリネットの書換によって良い非決定性を導入する。

【0101】（6） ステップA5：テスト・デバッグ  
良い非決定性が導入された超逐次プログラムに対して、ステップA3と同様に、テスト実行装置15でテストを行い、バグがある場合は、修正装置9でテスト・デバッグを行う。また、プログラムの修正が同期命令等の並行構造の変更にも及ぶ場合は、ステップA1に戻り、プログラム作成装置で修正を行い、再度セクションの設定及び逐次化を行う。ここで、プログラム6で並行化が導入された部分に関しては、並行に実行を行う。この場合において、超逐次プログラムの逐次化情報がベトリネット

タを用いて、トークン(token)のあるブレースに対応するセクションを実行することで、超逐次プログラムのテスト実行が実現できる。

【0102】（7） ステップA6：部分超逐次プログラムに対する良い非決定性の導入・拡大

ステップA5でテスト・デバッグが行われた超逐次プログラムに対して、良い非決定性に関する情報を追加する。ステップA5～ステップA6を所定の回数繰り返し、良い非決定性を徐々に拡大していく。更に良い非決定性を導入する必要がある場合は、ステップA5と同様に非決定性導入装置17で並行性を追加し、ステップA6に戻る。必要がなければ、ステップB2に進む。

【0103】（8） ステップB2：自動並行化

設計者による良い非決定性の導入が終わった超逐次プログラムに対して、解析装置10により抽出された超逐次プログラムの解析情報12により、並行可能な部分を自動抽出し、並行化装置18で並行化ルールを用いて超逐次プログラムの並行性を自動拡大する。並行化ルールは、例えば、以下のようにになっている。プログラム情報から、解析装置10によってデータ依存関係及び制御依存関係に基づく先行制約を抽出し、先行制約のないセクションに関しては並行化できる。この並行化処理は、予め定められた並行化ルールを適用することにより行うことができる。並行化ルールの一例としては、図12に示すようなベトリネットによる逐次化情報の書換ルールがある。データ依存関係及び制御依存関係に関しては、公知である。

【0104】（9） ステップB3：並行プログラムの作成

並行化装置18において、自動的に並行性を拡大した超逐次プログラムに対して、超逐次プログラムのプログラム情報をソースコードに反映させた並行プログラム15を生成する。

【0105】通常、逐次プログラムに比べて、並行プログラムのテスト・デバッグは非常に困難な作業である。これは、並行性によるプログラムの非決定により、あるタイミングによっては、プログラムが設計者の意図しない挙動を示すからである。並行プログラムのテスト・デバッグでは、設計者の意図しない並行性（悪い非決定性）によるバグを1つ1つテストで発見して取り除いている。しかし、この方法では、バグを完全に排除するのは非常に困難であり、更に労力を要する。

【0106】本発明の超逐次プログラミングでは、並行プログラムをまず逐次化し、それに対して設計者が意図する並行性を徐々に導入し、最終的に自動的に並行化できる部分を並行化し、並行プログラムを復元する。

【0107】すなわち、本発明の超逐次プログラミングは、並行プログラムから悪い非決定性を除くのではなく、逐次プログラムに良い非決定性を導入する。このように、超逐次プログラミングでは、逐次プログラムから

ボトムアップ的に並行プログラムを作成するので、予期せぬタイミングで発生するバグが入る余地がなく、非常に信頼性の高いプログラムを作成することができる。また、テスト・デバッグがはるかに容易になる。逐次プログラミングでは、逐次化による性能劣化等が懸念されるが、スーパーコンピュータ等の分野におけるFORTRAN等の逐次プログラムの自動並行化技術が利用できるので、多くの場合では実用上問題がない。実施例2の基本的な構成及び動作を説明したが、以下に、実施例2の具体例を示す。

#### (a) 第1具体例

図13は、並行プログラムの例を示す。ここで、P1とP2は並行に動くプロセスである。また、P1とP2は、共有メモリMをアクセスしている。

【0108】(1) ステップA1:モデル化  
並行プログラムPを図14のように記述する。

【0109】(2) ステップB1:セクションの設定  
この場合は、各命令が1つのセクションを形成すると考える。簡単のため、セクションIDを命令そのものとする。

【0110】(3) ステップA2:逐次化  
プロセス優先度を導入することによる逐次化を行う。具体的には、 $P1 \gg P2$  ( $P1$ は $P2$ より優先する)とする。この時、逐次化されたセクションの実行順序は、以下のようになる。

```
init1→read1→write1→read2→
write2
```

逐次化により生成された超逐次プログラムは、それぞれ図15に示すように、セクション情報、プログラム構造情報、超逐次化情報から構成される。ここで、逐次化情報は、上記の実行順序をベトリネットで表現したものである。

【0111】(4) ステップA3:超逐次プログラムに対するテスト・デバッグ

超逐次プログラムを実行し、バグがあればセクションの各セクションのソースコード、或いは、オリジナルの並行プログラムを修正する。ここでは、バグはなかったとする。

【0112】(5) ステップA4:超逐次プログラムに対する良い非決定性の導入

逐次化情報のベトリネットを表示装置で表示し、逐次関係を切断することにより、並行化を行う。ここでは、 $write1$ と $read2$ の逐次関係を切断することにする(図16(a))。 $write1$ と $read2$ はプログラム構造情報での実行順序関係はないので、切断可能である。ここで、どの逐次関係を切断すべきかに関するガイダンス情報を後述する解析情報20に基づき提供することもできる。

【0113】(6) ステップA5:テスト・デバッグ  
並行性の導入された超逐次プログラムをテスト実行す

る。ここでは、

```
init1→read1→write1→read2→
write2
init1→read1→read2→write1→
write2
```

のような実行が可能である。実行の結果、バグがあれば修正する。ここでは、バグはなかったとする。

【0114】(7) ステップB2:自動並行化  
超逐次プログラムの逐次化情報とプログラム情報のデータ依存関係から、解析情報20により、セクション間の先行制約が得られる。この先行制約が解析情報20である。先行制約とは、データ依存関係にあるセクション間の実行順序の制約である。すなわち、データ依存関係にあるセクション間は、実行順序によって計算結果が変わり得るので、逐次化情報で定められた順序を保持する必要がある。ここでの先行制約は以下の3つである。

```
init1→read2
init1→write2
read1→write2
```

例えば、 $P1$ が $read1$ で読み込む値は、 $P2$ の $write2$ が $read1$ の前で起こるか後で起こるかにによって影響を受ける。逐次情報では、 $read1 \rightarrow write2$ であるので、これが先行制約となる。以上の先行制約がないセクションに関しては、並行化が可能であり、並行化ルールにより自動並行化できる。ここでは、 $read1$ と $read2$ には先行制約がないため、並行化ルール1(図12)を適用できた(図16(b))。

【0115】(8) ステップB3:並行プログラムの作成

自動並行化を行った逐次プログラミングから、並行プログラムのソースコードを生成する。この例では、図10における逐次化情報1と逐次化情報2を実現する同期命令( $send$ ,  $wait$ )をソースコードに埋め込む。その他の逐次化情報は、オリジナルの並行プログラムが持っている逐次化情報(実行順序)である。変換されたプログラムは図17のようになる。この並行プログラムは、図13のような構造の並列計算機上で実行される。

#### (b) 第2具体例

第2具体例では、並行プログラムが同期命令、ループ構造、条件分岐を持つ場合の超逐次プログラミングの手順を示す。

【0116】(1) ステップA1:モデル化及びステップB2:セクションの設定は省略し、図18のようなセクションの実行順序(プログラム構造情報)を持つ並行プログラムが与えられたとする。ここで、同期命令( $send$ ,  $wait$ )とデータ依存関係( $S13$ と $S22$ )、ループ構造及び条件分岐があることに注目されたい。

【0117】(2) ステップA2:逐次化  
プロセスに優先度( $P1$ が $P2$ より優先度が高い;すな

わち、 $P1 > P2$ )を導入することによる逐次化を行う。逐次化の結果は図19のようになる。すなわち、超逐次プログラムは、それぞれ図18及び図19のプログラム構造情報と逐次化情報を持つ(セクション情報は省略)。ここで、同期命令によって実行されるプロセスが切り替わっていることに注目されたい。例えば、wait命令によって、P1のS12の実行の後にP2のS21が実行されている。また、ループ及び分岐構造もベトリネットで表現されている。

【0118】(3) ステップA3: 超逐次プログラムに対するテスト・デバッグ

超逐次プログラムを実行し、バグがあればセクションの各セクションのソースコード、或いは、オリジナルの並行プログラムを修正する。ここでは、バグはなかったとする。

【0119】(4) ステップA4: 超逐次プログラムに対する良い非決定性の導入

逐次化情報のベトリネットを表示装置で表示し、逐次関係を切断することにより、並行化を行う。この例では、明示的には、良い非決定性を入れなかったとする。

【0120】(5) ステップB2: 自動並行化

超逐次プログラムの逐次化情報とプログラム情報のデータ依存関係から、セクション間の先行制約が得られる。ここでは、先行制約としては、 $S22 \rightarrow S13$ だけである。ここで、ループにおいては、先行関係にもループが生じることがある。ここでは、1回のループにおける先行関係のみに注目する。先行制約がないセクションに関しては、並行化が可能であり、並行化ルールにより自動並行化できる。この例では、先行関係は $S22 \rightarrow S13$ だけであり、図12の並行化ルール1と並行化ルール2により、最終的に図20が生成できる。

【0121】(6) ステップB3: 並行プログラムの変換

第1具体例と同様に、残った逐次化情報に関して同期命令を埋め込むことによって並行プログラムが生成できる。

【0122】(実施例3) 図21は実施例3に係る並行プログラム作成支援装置の概略構成を示す図であり、図22は、本実施例に係る並行プログラム作成方法の概略手順を示すフローチャートである。

【0123】本実施例では、テスト実行装置401とテストケース記憶部402によって逐次化装置12が構成される。この逐次化装置12では、今までの実施例と異なり特別に逐次化ルールはなく、逐次化はランダムに行われる。すなわち、ランダムに実行したときのログを超逐次プログラムとみなす。以下、本実施例を具体的に説明する。

【0124】本実施例における並行プログラムの作成は、以下のような手順によって実現される。ここで、第1並行プログラムとして全く独立に動く2つのプロセス

P1とP2から構成される簡単な並行プログラムを考える。

【0125】(1) ステップA1: モデル化

対象の並行システムに対し、並行性を用いた自然なモデル化を行う。また、各プロセス構造を決定する。更に、該プロセス内を並行プログラム等を用いたプログラミングにより、並行構造を有する並行プログラムをソースプログラムとして記述する。このソースプログラムには、バグが潜在的に存在する可能性がある。ここでは、2つのプロセスP1とP2を図23に示すようにプログラミングする。

【0126】(2) ステップC1: テスト

テスト実行装置401により、テストケース記憶部402からテストケースを引き出し、CPファイル記憶部11からの第1並行プログラムを実行し、実行結果を出力装置5に表示してランダムにテストを行う。ここでは、実行ログが、

log1 = job11 → job12 → job21 → job22

であったとする。

【0127】(3) ステップC2: バグ判定

ステップC1のテストの結果、バグがなければステップC4に進み、バグがあれば、その実行ログを実行ログ記憶部403に保存してステップC3に進む。ここで、もしバグがあればステップC3に進むが、ここではバグがなかったとしてステップC4に進む。

【0128】(4) ステップC3: テスト・デバッグ  
デバッグ装置16により、実行ログ記憶部403に保存された実行ログに基づくバグの発見を出力装置5を用いて行い、CPファイル記憶部11に格納されている第1並行プログラムを修正してバグを除去する。テスト・デバッグが完了したら、ステップC1に戻る。

【0129】(5) ステップC4: 実行ログの蓄積  
ステップC1のテストの結果、ステップC2でバグがないと判定されれば、実行ログlog1を実行ログデータベース404に蓄積する。実行ログは、超逐次的な実行系列であり、超逐次プログラムの特殊な形態とみなすことができる。

【0130】(6) ステップC5: 残テストケースの有無判定

テストケース記憶部402にテストケースが残っているかどうかを判定し、残っていればステップC1に戻ってテストを続行し、テストケースが残っていなければステップA7に進む。本ステップでは、別のテストケースも試みるためにステップC1に戻ってテストを続行する。この2回目のテストで、次に示す実行ログlog2が実行ログデータベース404に保存されたとする。

log2 = job11 → job21 → job12 → job22

(7) ステップA7: 並行プログラムの生成



CPファイル記憶部11に格納されている第1並行プログラムと実行ログデータベース404に格納されている実行ログから、テストで通過したパスだけしか実行しないような並行プログラムを並行化装置18により生成し、第2CPファイル記憶部19に格納する。

【0131】本ステップでは、実行ログデータベース404に保存された2つの実行ログlog1、log2だけを実行するような並行プログラムを以下の手順で生成する。

(A) 全ての実行ログをマージしてグローバル状態遷移システムGTS(Global Transition System)を生成する(図24(a))。

(B) GTSを2つのプロセスP1、P2に射影する。この時、相手のプロセスの挙動を認識し、同期を取るための同期命令(job11-ok、job21-ok、job12-ok、job22-ok)が埋め込まれる。生成されたプログラムをP1'、P2'とする(図24(b))。

(C) P1'、P2'の同期命令には冗長性があるため、この冗長性を除去する。この冗長性を除去した後の最終的なプロセスをP1''、P2''とする(図24(c))。

【0132】上記の手順により、2つのプロセスP1''、P2''から構成される並行プログラムCP''が生成できる。この並行プログラムCP''では、テストで正しく動くことを確認した実行log1、log2は可能だが、テストしていないケース、例えば

log3=job21→job11→job12→job22

は実行されることはない。

【0133】上記のように、テストしていないケースは実行されることはないので、非常に安全なプログラムであるといえる。

【0134】実施例3によれば、逐次プログラミングと同程度の困難さで並行プログラムのプログラミングができ、ユーザはテストを効率的にできるばかりではなく、テストしない部分は動かないので、非常に高い信頼性が達成できる副次的効果も期待できる。

【0135】(実施例4)図25は、実施例4における並行プログラム作成方法の概略手順を示すフローチャートである。

【0136】(1) ステップA1:モデル化

(2) ステップA2:逐次化

(3) ステップA3:超逐次プログラムのテスト・デバッグ

以上のステップA1～A3は、図3に示したステップA1～A3と全く同様であるため、説明を省略する。

【0137】(4) ステップA4:超逐次プログラムに対する良い非決定性部分の導入

ステップA3でテスト・デバッグが行われた超逐次プロ

グラムに対して、ユーザが非決定性導入装置17により、非決定的に動作する部分(これを「良い非決定性部分」という)を指定する。非決定性導入装置17により指定された良い非決定性部分は、超逐次プログラムに良い非決定性に関する情報(並行性に関する情報)として反映される。これにより超逐次プログラムは、良い非決定性部分を持つ部分超逐次プログラムとなる。良い非決定性部分の指定方法は後述する。

【0138】(5) ステップA5:部分超逐次プログラムのテスト・デバッグ

ステップA4で得られた部分超逐次プログラムに対して、テスト・デバッグを行う。すなわち、ステップA4で良い非決定性部分が導入された部分超逐次プログラムをテスト・デバッグする。ここでは、部分超逐次プログラムを実行形式に変換して実行する際に、ステップA4において導入された良い非決定性部分のみが並行構造の実行形式に、導入されなかった部分は元の逐次構造のまま実行形式に変換され、実行される。そして、この実行結果に基づいて、超逐次プログラムに対してテスト・デバッグする。変換された実行形式のプログラムが正しく動作しない場合は、変換前の逐次プログラムにおいて導入した良い非決定性部分は本来、非決定的に動作させるべき部分ではないとみなして、並行化の解除を行う。

(6) ステップA6:超逐次プログラムにおける良い非決定性部分の拡大

ステップA5でテスト・デバッグが行われた部分超逐次プログラムに対して、良い非決定性に関する情報を追加する。ステップA4～ステップA6を所定の回数繰り返して、良い非決定性部分を徐々に拡大していく。

【0139】(7) ステップA7:並行化コンパイル  
メタレベルのデフォルト逐次性で管理されている部分超逐次プログラムのデフォルト逐次性を解除する(例えば、ソースプログラム自体に逐次情報及び並行情報を直接埋め込む)。また、この時、部分超逐次プログラムのうちの良い非決定性に関する情報が導入されなかった部分について、可能な限り、無害な非決定性部分を抽出し、その部分を並行化することを試みて、部分超逐次プログラムの並行性を高める。例えば、良い非決定性に関する情報が導入されなかった部分について各プロセス間の依存関係を解析し、依存関係がないと解析された部分に対応するソースプログラムの部分の逐次性を解除する。なお、無害な非決定性部分の抽出は、ステップA2で逐次化する際に解析された並行情報を参照することにより行われる。

【0140】次に、本実施例に係る並行プログラム作成方法及び作成支援装置の詳細について説明する。

【0141】本実施例は、出力装置5上に表示された超逐次グラフを用いることにより、非決定性部分の導入を、視覚的に行うことができるようにしたことを特徴とする。超逐次グラフとは、並行構造を有する並行プロ

ラムに対してデフォルト逐次性を導入した超逐次プログラムのプロセス処理順序を表した遷移グラフをいうものとする。この超逐次グラフの表示は、逐次化情報に基づいて行われる。逐次化情報は、逐次化装置12により並行プログラムが超逐次プログラムに変換される際に生成される。

【0142】より具体的には、逐次化情報とは、所定のフィールドを有するプロセステーブルと呼ばれるデータ列群である。図26は、プロセステーブルの構造を示す図である。

【0143】図26において、名前フィールドF1は、個々のプロセスを識別するための名前を保持する。ポインタフィールドF2は、当該プロセスから呼び出されるプロセス（以下「被呼出プロセス」という）の名前リストを格納する被呼出プロセスリストテーブル（図示しない）のポインタを保持する。この被呼出プロセスリストテーブルは、別に設けられている。優先順位フィールドF3は、プロセスの優先順位を保持する。このプロセスの優先順位は、もとの並行構造を有するプログラムに基づき、ある処理時点において複数のプロセスが実行可能な場合に、逐次化装置12によって決定され、その結果が当該フィールドに記述される。優先順位バッファフィールドF4は、後述するように非決定性導入装置17により、その値が変更される。このため、優先順位バッファフィールドF4は、変更前の優先順位を保持することにより、変更前の状態に戻すことを可能とする。グループ化情報フィールドF5は、特定のノード群がグループ化された場合に、各グループ間を識別する情報を保持し、グループ化されたノード群に対応するプロセス群のうち、最初に実行されるプロセスの名前が書き込まれる。

【0144】本実施例における逐次化装置12について説明する。

【0145】図27に示すような並行プログラムが逐次化装置12に読み込まれたとする。逐次化装置12は、まず並行プログラムの並行構造を解析する。図27に示す並行プログラムは、概念的には図28のような並行構造を有するものであると解析される。この解析は、例えば、木構造探索アルゴリズム等を用いることにより実現できる。そして、逐次化装置12はその解析結果をプロセステーブルに書込む。具体的には、逐次化装置12は、図26に示したプロセステーブルの優先順位フィールドF3に優先順位を書込むことによってデフォルト逐次性を導入し、超逐次プログラムに変換する。すなわち、図28において、プロセスBの処理の後、プロセスCとプロセスDとのいずれが処理されるかは、その時点におけるシステム環境等により異なる。このような場合に、逐次化装置12は、所定の規則（逐次化ルール）により優先順位を一意に決定する。なお、所定の規則とは、例えば、プロセスの読み込み順に従うとする規則や

乱数により決定するという規則等があげられるが、種々の事情に応じてユーザが設定することができるものとする。なお、本実施例における超逐次プログラムは、概念的には、ソースプログラムとそれをメタレベルで管理する逐次化情報とにより構成されることになる。

【0146】例えば、上記の例において、所定の規則によりプロセスCの方がプロセスDよりも優先的に処理するものと決定されると、プロセステーブルにおけるプロセスCの優先順位フィールドF3には「1」が書き込まれ、プロセステーブルにおけるプロセスDの優先順位フィールドF3には「2」が書き込まれる。なお、優先順位を決定する必要がないプロセスについては、優先順位フィールドF3は「0」のままである。この優先順位の決定は、階層レベルが等しいプロセス間で行われる。例えば、図28におけるプロセスC、G、IとプロセスD、H、Jとは、同じレベルのプロセスであるが、その中のプロセスEとプロセスFとは一階層下のレベルであるとされる。

【0147】このようにして逐次化装置12は、並行プログラムの並行構造を解析して逐次化情報（プロセステーブル）を生成し、これをファイルに記録する。

【0148】図29は、逐次化装置12によって並行プログラムの構造を解析した結果作成されたプロセステーブルの内容を示す図である。すなわち、図29はプロセスAからプロセスKまでに対応する各レコードにおける各フィールドの内容を示す。より具体的には、プロセスAの次に呼び出されるプロセスはプロセスBであり、プロセスAの優先順位は0であることを示す。また、例えば、プロセスBでは次に呼び出されるプロセスとしてプロセスC又はプロセスDであることを示しており、プロセスCとプロセスDとは本来非決定的であるということの意味している。そして、このプロセスCとプロセスDとは逐次化装置12によりそれぞれ優先順位「1」と「2」とが割り当てられ、プロセスCがプロセスDよりも優先的に呼び出される（実行される）ことを意味している。

【0149】良い非決定性に関する情報の導入（つまり、良い非決定性部分の指定方法）について説明する。良い非決定性部分の導入は、図2に示した非決定性導入装置17を用いたユーザによる対話的操作により行われる。すなわち、この非決定性装置17は、出力装置5上に超逐次グラフを表示し、ユーザは、この超逐次グラフに対して入力装置4を用いて、良い非決定性部分の導入・解除を行う。

【0150】図30は、超逐次グラフの一例を示す図である。図30において、各ノードは各プロセスを表し、破線矢印で示されたアークはもとの並行プログラムの並行構造を表している。また、実線矢印は逐次化装置12によりデフォルト逐次性が導入されることにより決定されたプロセスの逐次構造を表し、その逐次構造に従って

プロセスの実行順序が連番で付されている。つまり、もとの並行プログラムにおいては、プロセスC→G→IとプロセスD→E→F→H→Jとは、本来並行処理されるように記述されていることを表しているが、超逐次プログラムにおいてはプロセスC→G→Iを一連に処理した後、プロセスD以降を処理することを表している。

【0151】ユーザは、このような超逐次グラフを出力装置5により視覚的に把握することができる。更に、ユーザは、並行プログラムが逐次化装置12により逐次化され、逐次化情報が生成された後は、超逐次グラフを表示する旨の指示を与えることにより、いつでも出力装置5により視覚的に把握することができる。本実施例では、超逐次グラフの表示は、非決定性導入装置17の超逐次グラフ表示制御部によって行われる。この超逐次グラフ表示制御部は、ユーザにより超逐次グラフを表示する旨の指示を受けると、所定の内蔵プログラムに従って超逐次グラフの表示を開始する。

【0152】図31は、超逐次グラフ表示制御部の構成を示す機能ブロック図である。超逐次グラフ表示制御部は、逐次化情報記憶部31に格納されている逐次化情報に従って、各プロセスに対応するノードと、ノード間の呼び出し関係を示す並行構造及び逐次構造に対するアークを決定し、これらを画像データ生成部35により出力装置5上に表示するために必要な画像データに変換する。

【0153】逐次化情報読み込み部32は、逐次化情報記憶部31から逐次化情報を読み込み、並行構造解析部33及び逐次構造解析部34に送出する。

【0154】並行構造解析部33及び逐次構造解析部34は、逐次化情報を参照して並行構造を解析する。具体的には、並行構造解析部33は、ポインタフィールドF2に指示された被呼出プロセスリストテーブルを参照することにより、名前フィールドF1に記述されたプロセスによって呼出され得るプロセス名の間の呼出関係を特定する。逐次構造解析部34は、これらのフィールドに加え、更に優先順位フィールドF3に記述された優先順位を参照して逐次構造を特定する。これら並行構造解析部33及び逐次構造解析部34による解析結果は、画像データ生成部35に送出される。画像データ生成部35は、入力されたこれらの解析結果に基づいて、各プロセスに対応するノードを表示するための画像データと、該プロセス間の呼出関係を接続するための2つのアーク（つまり、並行構造に対するアーク及び逐次構造に対応するアーク）を表示するための画像データを生成し、出力装置5に送出する。出力装置5は、これらの画像データに基づいて遷移グラフ（即ち、超逐次グラフ）を表示する。

【0155】なお、上記超逐次グラフの表示に際し、並行構造解析部33により解析された並行構造の各ノード間の接続関係に、従属的に逐次構造解析部34により解

析された逐次構造の各ノード間の接続関係を付加することにより画像データを生成することが望ましい。換言すれば、超逐次グラフは、もとの並行プログラムの並行構造を保持した超逐次グラフであることが望ましい。これにより、ユーザは、もとの並行プログラムが本来持つ並行構造を意識しつつ、逐次化された処理の流れ（プロセス）に対して、良い非決定性部分を導入することができる。

【0156】出力装置5に超逐次グラフが表示されると、図2の非決定性導入装置17はユーザによる良い非決定性部分の入力待ちの状態になる。（部分）超逐次プログラムのテスト・デバッグ開始当初は、非決定性部分が導入されていないので、逐次構造に対するアークは出力装置5上の全てのノードに対して設けられており、並行化情報は表示されていない。ここで、並行化情報とは、良い非決定性部分が導入されたノードに対して与えられる情報であり、良い非決定性部分が導入されたか否かにより画面上で視覚的に区別されて（例えば陰影の濃度、色分け等）表示される。また、この並行化情報は、階層化された良い非決定性部分についても視覚的に区別されて表示される。

【0157】ユーザによる良い非決定性部分の導入は、入力装置4を用いて行われる。具体的な良い非決定性の導入方法については後述するが、略説すれば、入力装置4を用いてノード間の逐次構造の接続関係（アーク）を断ち切ったり、接続したりすることにより行われ、その結果は、逐次化情報変更部36により逐次化情報記憶部31の内容が変更される。変更された内容は、再び逐次化情報読み込み部により読み込まれ、出力装置5上に表示されることになる。

【0158】良い非決定性部分の導入の具体的操作例について説明する。

【0159】図32は、良い非決定性部分の導入方法を説明するための図である。

【0160】複数のプロセス間に対して良い非決定性を導入する場合、良い非決定性の導入モードにおいて、該プロセスに対応する所望のノードを続けて指定することにより行われる。良い非決定性の導入モードは、例えば、指示操作メニュー等により選択され、該モードに移行する。

【0161】図32において、プロセスEとFとの並行化を許容する場合、入力装置4により操作されるカーソルPでノードE、Fを続けて指定する。この指定されたノードは、指定されていない他のノードと視覚的に区別できるように、陰影が付される。対象とするノード全てについて指定が完了した場合、ユーザは完了した旨を非決定性導入装置17に与える。これにより、逐次化情報変更部36は、逐次化情報記憶部31内のプロセステーブルにおける対応するプロセスの優先順位フィールドF3の現在の値を優先順位バッファフィールドF4に対比

させるとともに、優先順位フィールドF3の値を「0」に変更する。

【0162】つまり、本実施例では、プロセスE及びFの優先順位フィールドF3の値「1」及び「2」がそれぞれ優先順位バッファフィールドF4に対比されるとともに、それぞれの優先順位フィールドF3の値は「0」に変更される。逐次化情報変更部36による逐次化情報記憶部31の更新後、再度、出力装置5上に超逐次グラフが表示される。この時、プロセスEとFとの間の優先順位はないため、それに対応する逐次構造のアーキは表示されない。つまり、出力装置5上では、プロセスEとFとは、並行に動作することが確認される。

【0163】図33は、上記の場合において、良い非決定部分が導入された超逐次グラフの一例を示す図である。

【0164】上記のようにして良い非決定部分が導入されたプログラム（部分超逐次プログラム）は、テスト実行装置15により実行形式に変換され、テスト実行される。ユーザは、テスト実行装置15による実行結果に基づいて、プログラムが正しく動作するか否かを確認する。この動作確認は、並行性に基づく全てのパスについて実行する必要がある。テスト実行装置15が正しく動作しなかった場合、良い非決定部分を導入した部分は、本来並行化を許容する部分ではないとみなすことができるため、ユーザは並行化を解除する指定をする。

【0165】並行化の解除は、非決定性部分解除モードにおいて、良い非決定性部分を導入する場合と同様に指定することにより行われる。これにより、逐次化情報変更部36は、逐次化情報記憶部31内のプロセステーブルにおける優先順位バッファフィールドF4に格納されている値を優先順位フィールドF3に戻す。

【0166】本実施例では、良い非決定性部分の導入に際し、階層レベルごとにグループ化することも可能である。例えば、プロセスC、G、Iのグループ（グループ1という）とプロセスD、H、Jのグループ（グループ2という）とは同一の階層レベルであるので、これらは互いに並行動作しうる。従って、個々のプロセスについて操作したのでは操作性に欠けるため、これらをグループ化することが有効である。このグループ化は、グループ化モードを選択することにより行われる。具体的には、グループ化モードにおいて、対象とするノードを順次選択し、完了した旨を与えることにより、対象となるノードがグループ化されて1つの新たなノードとして表示される。

【0167】図34は、グループ化されたノードを持つ超逐次グラフの一例を示す図である。

【0168】図34において、グループ1及び2のノードは、グループ化されていない他のノードと視覚的に区別できるように、例えば、楕円形で表示されている。更に、グループ2は、更に下位階層（プロセスE及びF）

を持っているため、例えば、影付きで表示される。このような超逐次グラフに対し、ユーザは、良い非決定性部分を指定することができる。なお、このようなグループ化された超逐次グラフに対して良い非決定性部分を導入した場合であっても、先頭のプロセス（プロセスC及びD）の優先順位フィールドF3を変更するだけでよい。

【0169】超逐次グラフのノードに対して良い非決定性部分を導入する場合には、上述したように所望のノードを入力装置4により直接指定することにより行われるが、このような方法ではなく、特定のノード群を囲い込むことによって、又は指定領域が特定のノード群に掛かることによって指定されたとするようにしてもよい。なお、プロセスが多数ありそれに対応するノードが多数あるため、出力装置5上で確認することが困難な場合も想定できる。このような場合は、グループ化されたノード群のみを表示することにより対処することができる。

【0170】本実施例では更に、逐次化装置12により決定されたプロセス間の優先順位を変更することもできる。つまり、優先順位変更モードにおいて、所望のノードを指定することにより、対象とされたノード間の優先順位を変更する。

【0171】図32の超逐次グラフに対し、優先順位変更モードにおいて、プロセスEとFとの優先順位を変更するとする。ユーザは、上記のように入力装置4により、ノードE及びFとを指定し、指定完了の旨を非決定性導入装置17に与える。これにより、逐次化情報変更部36は、逐次化情報記憶部31内のプロセステーブルにおけるプロセスEの優先順位フィールドF3の値を「2」、プロセスFの優先順位フィールドF3の値を「1」と変更する。これにより、出力装置5上には、新たな超逐次グラフが表示されることとなる。図35は、この時の超逐次グラフの一例を示す図である。図35に示すように、優先順位変更後は、逐次構造に対するアーキが変更されることになる。

【0172】このようなプロセスの優先順位の変更は、ユーザの意図したプロセス間の動作が保証されるため、良い非決定性部分の導入に有効に役立てることができる。すなわち、まず、ユーザは図32の超逐次プログラムについてテスト実行装置15で実行形式に変換し、テスト実行を行う。次に、ユーザは、優先順位変更後の図35の超逐次グラフについて同様にテスト実行を行う。これにより、両方のテスト実行結果がユーザの意図したものであることが確認できれば、プロセスEとプロセスFは、どちらを先に実行しても良いとみなせるので、良い非決定性部分の導入を行うことができる。

【0173】なお、本実施例では、2プロセス間の非決定性の導入について説明したが、3プロセス以上の場合も同様である。

【0174】図36は、3プロセス間の良い非決定性部分の導入を説明するための図である。並行構造のプロセ

スB、C、D間において、その全てについて並行化を許容するようにすることもできるが、図36においては、プロセスBとCとの間のみ並行化を許容されている場合を示す。このような良い非決定性部分の導入により、超逐次グラフは図37のようになる。つまり、図37に示す超逐次グラフは、プロセスB及びCの両方が実行された後に、プロセスDが実行されることを示す。上記のように本実施例によれば、出力装置5上に表示した超逐次グラフによって並行化情報及び逐次化情報をユーザに対して同時に提示するので、ユーザはもとの並行構造を考慮しつつ、良い非決定性部分の指定をすることができるようになる。また、並行プログラム記述レベルにおける良い非決定性部分の指定・解除ではなく、超逐次グラフに対して良い非決定性部分を指定・解除をすることで、高度な並行プログラミング技術を必要とすることなく、容易に並行プログラムの開発をすることができるようになる。

【0175】(実施例5) 図38は、本実施例に係る並行プログラム生成作成支援装置のブロック図であり、特に図2におけるHSPファイル記憶部14と非決定性導入装置17について詳しく示す。

【0176】図38において、CPファイル記憶部11に格納されている第1並行プログラムは、逐次化装置12により先の実施例のように逐次化され、得られた超逐次プログラムはHSPファイル記憶部14内の逐次化プロセスリスト記憶部51に格納される。なお、超逐次プログラムは、図38には図示していないデバッグ装置により、逐次化プロセスリスト記憶部51に格納される時には既に逐次化された状態でのバグが取り去られている。

【0177】ここで生成された超逐次プログラム(逐次化プロセス)は、フィールドデータ生成部53によりフィールドデータに変換され、フィールドデータ記憶部52に保存される。これら逐次化プロセスリスト記憶部51とフィールドデータ記憶部52の情報は、中間ファイルたるHSPファイル記憶部14に保存される。フィールドデータ記憶部52からのフィールドデータは、フィールドチューニング部54を介してフィールドエディタ55での編集に供される。

【0178】非決定性導入装置17は、フィールドデータ生成部53、フィールドチューニング部54及びフィールドエディタ55で構成される。フィールドエディタ55で編集が終了したフィールドデータは、並行化装置18(フィールドデータ変換部)で修正された並行プログラムに変換され、第2CPファイル記憶部19に格納される。

【0179】図39は、図38におけるCPファイル記憶部11に格納されている第1並行プログラムの一例を示しており、説明の便宜上、意図的にバグを含ませている。ここでは、バグとしてプロセスP4とP5の並行実行は誤りであり、逆に、プロセスP2とP6は並行実行

が可能であるとし、正しい並行プログラムは後述する図66であるとする。図40は図39の第1並行プログラムで実行されるプロセスの流れをイメージした図であり、プロセスP4とP5、プロセスP7とP8がそれぞれ並行に実行される。図41は、図38における逐次化プロセスリスト記憶部51に格納されている逐次化プロセスの一例であり、逐次化装置12において図39の第1並行プログラムで並行な部分に、ある順番を付けることで逐次化されている。

【0180】また、図42は図41で実行されるプロセスの流れをイメージした図である。この例では偶然、P4、P5の順で逐次化されているので、この点に関するバグは取り除かれている。仮に、逐次化がP5、P4の順で行われていたなら、上記のバグが従来のデバック手法で検出されるはずであり、逐次化の時点で誤った並行化(悪い非決定性)を取り除くことができる。

【0181】図43は、図38におけるフィールドデータ生成部53で実行される処理の流れを示すフローチャートである。ここでは、1つのプロセスからもう1つのプロセスへ処理が進むことをある制約と遷移条件が存在する範囲(エリア)ととらえて、超逐次プログラム(逐次化プロセス)をフィールドデータに変換している。図43に示すように、ステップD1でスタートエリアを生成し、最後のプロセスが来るまでステップD3~D9を繰り返し実行する。そして、最後のプロセスが来たらステップE10~E12の処理を行って終了となる。図44は、図43に示した処理によって生成される一般的なエリアのデータ構造を示す。このエリアの接続で構成される全体構造としてフィールドが形成される。

【0182】エリアA(i)には以下のような情報が記述されている。

- (1) エリアA(i)にいる限りは、“constraint”に記述された制約が存在する。(例えば、P(x1)はP(x2)より先に実行される)
- (2) エリアA(i)から他のエリアへ移るための遷移条件は、“transition”に記述されている。(例えば、P(z1)が終了してA(j)へ遷移できる)
- (3) エリアA(i)から移ることのできるエリアは、A(j)である。
- (4) エリアA(i)へ移る以前の状態は、エリアA(k)である。

【0183】図45は、図43に示した処理により図41に記述されたプログラムをエリアの集合で構成されたフィールドに変換した様子を示す。

【0184】このフィールドは処理の順序関係の制約が最も強い。非決定性の導入は、この制約を変更したり、なくしたりすることで行なうことができる。つまり、フィールドチューニングを行うことで非決定性を導入し、並行化を図ることができる。なお、一般的には制約によるチューニングであるので、ユーザには制約と遷移条件

を表示する。

【0185】図46は、図38におけるフィールドチューニング部54で実行される処理のフローチャートである。

【0186】ステップE1で選択されたフィールドデータは、ステップE2においてそのエリア間の連結情報が解析されて視覚的にフィールドが生成され、フィールドエディタ55に表示される。ユーザは、フィールドエディタ55の画面を見ながらステップE3でコマンドを入力し、フィールドのチューニングを行うことができる。

【0187】ステップE5で編集中のフィールド保存でき、編集終了ならステップE17を経て終了する。なお、ステップE14～E16にて、編集による制約の矛盾を検出している。

【0188】ステップE9は制約の書き換え処理に関するサブルーチンである。また、ステップE11は制約の追加処理に関するサブルーチンである。そして、ステップE13は制約の削除処理に関するサブルーチンである。

【0189】図47は図46におけるステップE9の処理を行なうサブルーチンである。

【0190】ステップE9-1で編集された制約に関して、ステップE9-2で自明な制約の有無を検査し、もし自明な制約が存在すれば、制約の書き換えにエラーがあることをユーザにステップE9-9で伝え、ステップE9-10で書き換えを無効にする。もし、自明な制約が存在しなければ、ステップE9-4のようにフィールドを書き換える。そして、ステップE9-5～E9-8でフィールドチェックする。

【0191】図48は、図46におけるステップE9-4の制約の変更操作のアルゴリズムによって起こるフィールドの変化を示した図である。これは“before Pi Pj”を“before Pi Pj”に変更したときの様子を示しており、図48の(a)から(b)へとフィールドが変化する。ここでは、A(y)のエリアがエリアA(a)の“next”に接続され、それに伴って各エリアの属性が変化する。

【0192】図49は図46におけるステップE11の処理を行なうサブルーチンである。ステップE11-1で編集された制約に関して、ステップE11-2で自明な制約の有無を検査し、もし自明な制約が存在すれば、制約の書き換えにエラーがあることをユーザにステップE11-9で伝え、ステップE11-10で追加を無効にする。もし、自明な制約が存在しなければ、ステップE11-4のようにフィールドを書き換える。そして、ステップE11-5～E11-8でフィールドチェックする。

【0193】図50は図49におけるステップE11-4の制約の追加操作のアルゴリズムによって起こるフィールドの変化を示した図である。ここではエリアA(y)に“before Pj Pk”を追加したときの様子を示した

もので、図50(a)から(b)へとフィールドが変化する。ここでは、A(y)のエリアが新たにエリアA(x2)のnextに追加され、それに伴って各エリアの属性が変化する。

【0194】図51は図47におけるステップE13の処理を行なうサブルーチンである。

【0195】エリアには少なくとも1つの制約が必要であるので、ステップE13-1で指定したエリアA(y)の制約の個数を調べ、もし1つならステップE13-8でエラーメッセージを表示して終了する。もし、複数の制約があれば、ユーザの指定した制約を削除し、ステップE13-3のようにフィールドを書き換える。そして、ステップE13-4～E11-7でフィールドチェックし、自明な制約の発生を防ぐ。

【0196】図52は、図51におけるステップE13の制約の削除操作のアルゴリズムによって起こるフィールドの変化を示した図である。

【0197】これは“before Pj Pk”を削除したときの様子を示しており、図52(a)から(b)へとフィールドが変化する。ここでは、A(x2)のエリアがエリアA(z)の“pre”に接続され、それに伴って各エリアの属性が変化する。

【0198】図53はフィールドに矛盾を発生させる自明な制約を検出する処理であり、図47のE9-2、E9-5、図49のE11-2、E12-5、図51のE13-4で共通に実行されるサブルーチンである。ここでは、例えば“before A B”、“before B C”が存在するにも関わらず、“before A C”のような自明な制約を検出する。

【0199】図53においてE9-3～E9-2-6の繰り返しにより、自明制約が生成され、E9-2-7で制約集合との間に共通要素を探すことにより実現される。例えば、図54のような自明制約“before A D”を含んだフィールドを仮定した場合、この処理によって図55のテンポラリ集合(a)、テンポラリ集合(b)、テンポラリ集合(c)の順番で処理が実行され、共通要素“before A D”を検出することができる。

【0200】図56は、フィールドエディタ55の表示画面を示す。フィールドエディタ55では、フィールドの読み込み、保存、制約編集、表示モードの変更等をメニュー71を指定して行うことができ、四角で表現されたエリア72を選択すると、編集画面73が現れる。図57は、表示モードとして「2D表示モード」を選択した場合のフィールドの表示例を示す。このモードでは、平面的にフィールドが表示される。図58は、表示モードとして「3D表示モード」を選択した場合のフィールドの表示例を示す。このモードでは、立体的にフィールドが表示され、全体が大きくなったり、多くの並行処理部分がある場合にも、全体を見渡しやすくなることができる。

【0201】図59～図66は、図44に示したフィールドデータが図46のアルゴリズムでチューニングされていく様子を示す。

【0202】図59では、例えばユーザはP5とP6に因果関係がなく、P6はP2の後であればバグとなる悪い非決定的な動きはしない判断したとする。この場合、図59中に示されるように制約の書き換えるという編集を行うと、表示は図60のようになる。図60では、例えばP6とP4に因果関係はなく、P4はP3とのみ正しい順序が存在すると判断したとする。この場合、図60中に示されるようにある制約(“before P6 P4”)を削除する編集を行うと、表示は図61のようになる。以下同様に、ユーザがエリア内の制約を吟味し、徐々に編集を加えていくと、フィールドデータは図62→図63→…→図66のように変化する。

【0203】図67は、図38における並行化装置(フィールドデータ変換部)18で実行される処理を示すフローチャートである。図67に示されるように、並行化装置18ではフィールドデータ記憶部52からフィールドデータを読み込み(ステップF1)、エリア間のグラフ構造を解析してプロセスフローを生成し(ステップF2)、更にそのプロセスフローを解析して非決定的のみ導入された形で並行プログラムのソースを生成し(ステップF3)、第2CPファイル記憶部19に保存する(ステップF4)。

【0204】図68は、図66の状態となったフィールドを並行化装置18で解析して変換して得られた並行プログラムである。図68に示されるように、図39の第1並行プログラムで仮定したバグであるP4とP5の並行実行が直され、逆に、良い非決定性であるP3とP6の並行化が行われている。

【0205】また、図69は図68の並行プログラムで実行されるプロセスの流れをイメージした図である。

【0206】本実施例によれば、逐次化された並行プログラムのプロセスの流れを制約と遷移条件からなるフィールド(場)という概念として捉え、このフィールドをチューニングすることで良い非決定性を導入し、バグのない高品質な並行プログラムを効率的に生成することができる。また、本実施例では逐次化されたプログラムが制約を外すことによって並行化が進んでいくと、図形的に細長いフィールドだったものが縦方向に伸びていくので、直観的に理解しやすく、操作が容易となるという利点がある。

【0207】(実施例6)本実施例に係る並行プログラム作成方法及び作成支援装置では、テスト・デバッグは、超逐次プログラムに対して行う。また、良い非決定性に関する情報の導入も、超逐次プログラムに対して行う。

【0208】図70は、実施例6に係る並行プログラム作成支援装置の概略構成を示す図である。図70におい

て、CPファイル記憶部11に格納された第1並行プログラムは、ユーザによる入力装置4からの指示により引き出され、逐次化装置12に入力される。逐次化装置12に入力された第1並行プログラムは、逐次化ルール記憶部13に格納された逐次化ルールに従って完全超逐次プログラムHSPに変換され、複数の完全超逐次プログラムを格納可能なHSPファイル記憶部14に記録される。

【0209】この完全超逐次プログラムHSPに対して、テスト実行装置15でテストが行われる。ここで、バグがあればデバッグ装置16を用いて、HSPファイル記憶部14に格納されている完全超逐次プログラムHSPを修正する。テスト・デバッグが完了したならば、完全超逐次プログラムHSPをHSPファイル記憶部14に保存する。

【0210】次に、非決定性導入装置17で変更装置25を介して完全超逐次プログラムHSPの実行順序を変更し、別の完全超逐次プログラムHSPを生成する。この完全超逐次プログラムHSPに対して、同様のテスト・デバッグを繰り返しながら、完全超逐次プログラムHSPをHSPファイル記憶部14に蓄積していく。すなわち、良い非決定性を1つの部分超逐次プログラムとして表現する替わりに、複数の完全超逐次プログラムとして表現したものである。非決定性導入装置17による良い非決定性導入が完了したならば、最終的にHSPファイル記憶部14に蓄積された完全超逐次プログラムHSPの集合から、並行化装置18で並行プログラムが生成され、第2CPファイル記憶部19に格納される。

【0211】(実施例7)本実施例において並行プログラム作成支援装置の構成は、実施例6と同様であり(図70)、その並行プログラム作成手順が異なっているので、図示は省略する。図71は、本実施例における並行プログラム作成方法の概略手順を説明するためのフローチャートである。

【0212】(1) ステップA1:モデル化  
対象の並行システムに対し、並行性を用いた自然なモデル化を行う。また、各プロセス構造を決定する。更に、該プロセス内を並行プログラム等を用いたプログラミングにより、並行構造を有する並行プログラムをソースプログラムとして記述する。このソースプログラムには、バグが潜在的に存在する可能性がある。

(2) ステップA2:逐次化  
デフォルト逐次性を導入することによって、並行プログラムを逐次構造の超逐次プログラムに変換する。本実施例では、メタレベルにおいてデフォルト逐次性を導入する。ここで、メタレベルとはソースプログラムそのもののレベルについていうのではなく、ソースプログラムの実行を管理するレベルをいう。例えば、並行プログラムで記述されたソースプログラムを、それとは別に管理されるスケジューラによって逐次的に実行することを保障

したプログラムのソースプログラムに変換する。

【0213】(3) ステップA3：超逐次プログラムのテスト・デバッグ

ステップA2で変換された超逐次プログラムに対して、テスト・デバッグを行う。すなわち、テスト実行装置により逐次プログラムをテスト実行した結果に基づいて、デバッグ装置により超逐次プログラムからバグを除去する。ここでのテスト・デバッグは、逐次プログラムにおける通常のテスト・デバッグ方法と同様に行うことができる。超逐次プログラムが正常に動作することが保障されるまで、このテスト・デバッグを行う。

【0214】(4) ステップG1：並行模擬プログラム化

ステップA3でテスト・デバッグが行われた超逐次プログラムに対し、並行化の候補となるプロセス群を導入し、変更装置20によって複数の並行模擬動作系列を生成する。本実施例では、メタレベルにおいて並行模擬動作系列を導入する。

【0215】(5) ステップG2：並行模擬動作系列の実行・確認

ステップG1で得られた並行模擬動作系列をステップA3でテスト・デバッグが行われた超逐次プログラムを用いて実行し、実行結果の確認を行う。この実行は、プログラムが並行に動作する状況を超逐次プログラム上で模擬的に再現している。ユーザは、各動作結果が正しいか否かを並行模擬実行装置に記録する。

【0216】(6) ステップG3：良い非決定性の選択による部分逐次化

ステップG2の実行結果から、正しく実行された並行模擬動作系列を許容し、かつ実行の正しくない並行模擬動作系列を排除する非決定性を導入して、良い非決定性として登録する。ステップG1～ステップG3の処理を所定の回数繰り返し、良い非決定性の範囲を徐々に拡大していく。

【0217】(7) ステップA7：並行化コンパイル  
良い非決定性の導入された部分超逐次プログラムのうち、無害な非決定性部分を抽出し、その部分を並行化することで、部分超逐次プログラム全体を並行プログラムに変換する。すなわち、並行化に関する情報が導入されなかった部分に関しては、デフォルト逐次性を並行プログラムに反映させて（例えば、ソースプログラム自体に埋め込む）、メタレベルのデフォルト逐次性を解除する。

【0218】図72は、ユーザが想定していた並行プログラムモデルの例である。ここで、P1、P2、P3等はプログラムが実行される単位を表しており、これらをプロセスと呼ぶ。同図の例は、ユーザの意図においてプロセスP3、P4、P5は並行実行され、プロセスP7、P8、P9は逐次実行されることを表している。更に、プロセスP2、P3、P4、P5、P6からなるプ

ロセス群と、プロセスP7、P8、P9からなるプロセス群も並行実行可能である。

【0219】超逐次プログラムHSPは、逐次化装置12により並行モデルにおけるプロセスの前後関係を壊さない範囲で、全てのプロセスを一列に並べて得られる。

【0220】図73は、図72の並行プログラムから得られる超逐次プログラムHSPの実行系列をモデル化した例である。このように逐次化モデルでは、並行に実行されるプロセスP3、P4、P5等は任意の順で、場合によっては離れた位置に置かれるが、これらより先に実行されるべきプロセスP1、P2は、プロセスP3、P4、P5のいずれよりも前に置かれ、またこれらより後に実行されるべきプロセスP6、P10は、プロセスP3、P4、P5のいずれよりも後に置かれる。同様に、プロセスP2、P3、P4、P5、P6からなるプログラム群と、プロセスP7、P8、P9からなるプログラム群を逐次化した際の群の間の前後関係は任意で、場合によっては両方の群のプロセスが逐次モデル上に交互に表れることもある。

【0221】超逐次プログラムHSPは、ユーザによる入力装置4からの指示により、テスト実行装置15に入力され、テスト実行が行われる。テスト実行装置15は、テスト実行の結果を出力装置に提示する。ユーザは、このテスト実行の結果に基づいて超逐次プログラムHSPに対し、入力装置4を用いて所定のテスト・デバッグを行うことができる。また、ユーザは超逐次プログラムHSPに対して所定のテスト・デバッグを行った後、再度入力装置4からテスト実行の指示を与える。これにより、テスト・デバッグの行われた超逐次プログラムHSPは、テスト実行装置15に入力され、再度テスト実行が行われる。このテスト・デバッグは、プログラムHSPが正常に動作することを確認するまで繰り返し行われる。

【0222】テスト・デバッグにより正常に動作することを確認した時点で、非決定性入力装置17により超逐次プログラムを部分逐次プログラムへ変換する。まず、超逐次プログラムを複数の並行模擬プログラムへ変換し、並行模擬プログラムの実行結果より、良い非決定性に関する情報を蓄積する。この手順を図74に示す。この図74に示す手順に従い、図73の逐次モデルを並行模擬モデルへ変換し、部分逐次モデルを得るまでの過程の具体例を図75～図78に示す。

【0223】以下、この例の手順について説明する。

【0224】図75(a)ではポインティングデバイス等によって、並行模擬範囲としてP3、P4、P5が選ばれる(ステップH1)。更に、各々のプロセスが単独で並行化の単位に指定される(ステップH2)。図75では、並行化単位が色の違いによって区別される。超逐次プログラムでは、この並行模擬範囲の内部でプロセスはP3、P4、P5の順番で実行される。



【0225】図75(b)では、並行化単位の置き換えとしてP4、P5が置き換えられ(ステップH3)、一つの並行模擬動作系列が獲得される。この並行模擬動作系列では、並行模擬範囲の内部でプロセスはP3、P5、P4の順番で実行される。このプログラムを実行して実行結果が保証されると(ステップH4)、図75(c)のようなプログラムが獲得される(ステップH5)。この部分逐次プログラムでは、並行模擬範囲の内部でプロセスP3を実行した後にP4とP5が並行に実行される。

【0226】更に、同一範囲内での非決定性の拡大がユーザによって指示されると(ステップH6)、図76(a)では単独のプロセスP3と2つのプロセスの組P4&P5が置き換えられ(ステップH3)、もう一つの並行模擬動作系列が獲得される。この並行模擬動作系列では、並行模擬範囲の内部でプロセスはP4とP5が並行に実行された後にP3が実行される。このプログラムを実行して正しさが確認されると(ステップH4)、図76(b)のような非決定性を持つプログラムが獲得される(ステップH5)。この部分逐次プログラムでは、並行模擬範囲の内部でプロセスP3、P4、P5が並行に実行される。これによって、この並行模擬範囲内での非決定性は十分となるので、ステップH6からステップH7に進み、並行範囲の拡大がユーザによって指示されると(ステップH7)、図77~図78に示すように、ステップH1に戻って同様の処理を繰り返す。

【0227】良い非決定性に関する情報がインクリメンタルに導入され得られた部分逐次プログラムPHSPは、ユーザからの指示により、並行化装置18に入力される。この並行化装置18では、良い非決定性のみを持つ部分逐次プログラムPHSPを並行プログラムCPとして第2CPファイル記憶部19に記録する。ユーザは、このプログラムCPを出力装置5によって見ることができるとともに、最終的なテスト・デバッグを行うことができる。

【0228】本実施例によれば、超逐次プログラム上で並行プログラムの動作を十分に確認することができる。また、超逐次プログラムに対して並行化の候補となる範囲を指定することで、超逐次プログラムを段階的に並行プログラムへ変換することができる。更に、並行模擬動作系列に基づく逐次実行で正しく動作することが確認された非決定性のみを許容する並行性を導入することで、正しく動作する並行プログラムを得ることができる。これによって、並行プログラムのテスト・デバッグが更に容易となるという利点がある。

【0229】(実施例8)本実施例は、図1のように構成されたコンピュータシステムにおいて、図79に示すように各プロセッサ1-1~1-Nに少なくとも1個以上のプロセスA~Dが登録され、それぞれのプロセスは並行/並列に動作しており、互いにメッセージを交換し

ながら情報交換して処理を行うというプロセス実行環境において、並行プログラムの作成を行う例である。

【0230】本実施例では、並行プログラムのテストを超逐次プログラムに対して行うが、バグのあった場合はソースコードたる並行プログラムを修正せずに、逐次化ルールを修正することでテスト・デバッグを行う。

【0231】図80は、本実施例に係る並行プログラム作成支援装置の概略構成を示す図である。

【0232】図80において、CPファイル記憶部11に格納されている第1並行プログラムは、逐次化装置12において逐次化ルール記憶部13に格納された逐次化ルールに従って、メタレベルにおいてデフォルト逐次性が導入されることにより、超逐次プログラム(HSP)に変換され、HSPファイル記憶部14に記録される。ここで、逐次化ルールはソースプログラム(CPファイル)を実行装置22で実行させた際に得られる実行ログ情報となり、HSPファイル記憶部14はCPファイルと逐次化ルールである実行ログ情報の対からなる。ユーザは、この超逐次プログラムHSPを出力装置5によって見ることができる。この超逐次プログラムHSPは、ユーザによる入力装置4からの指示によってテスト実行装置15に入力され、テスト実行が行われる。

【0233】テスト実行装置15は、テスト実行の結果(実行ログ)を出力装置5に提示する。ユーザは、このテスト実行の結果に基づいて、逐次化ルール修正装置26を用いて逐次化ルール記憶部13に格納されている逐次化ルールを修正し、超逐次プログラムHSPに対し所定のテスト・デバッグを行うことができる。

【0234】ユーザは、このように逐次化ルールの修正により超逐次プログラムHSPに対して所定のテスト・デバッグを行った後、入力装置4から再度テスト実行の指示を与える。これにより、テスト・デバッグの行われた超逐次プログラムHSPは、テスト実行装置15に入力され、再度テスト実行が行われる。このテスト・デバッグは、超逐次プログラムHSPが正常に動作することを確認するまで繰り返し行われる。

【0235】そして、本実施例ではテスト・デバッグにより正常に動作することを確認した時点で、逐次化ルール修正装置26により逐次化ルール記憶部13に格納されている逐次化ルールが修正され、この修正の後の逐次化ルールが逐次化ルール記憶部13に新たに記録される。この逐次化ルールの修正によって、超逐次プログラムに対して良い非決定性に関する情報が部分的に導入される。超逐次プログラムHSPは部分超逐次プログラムPHSPに変換される。

【0236】次に、部分超逐次プログラムPHSPは、潜在的に存在する無害な並行性を顕在化させるために、並行化装置18にて部分超逐次プログラムPHSPにおける逐次化ルールを分割基準指定装置23からの指示に基づいて各プロセス毎に分割し、各プロセス毎の起動順

序制御を可能とする第2並行プログラム19に変換する。

【0237】図81は、本実施例に係る並行プログラム作成方法の概略手順を説明するための図である。

【0238】(1) ステップA1：モデル化  
対象の並行システムに対し、並行性を用いた自然なモデル化を行う。また、各プロセス構造を決定する。更に、該プロセス内を並行プログラム等を用いたプログラミングにより、並行構造を有する第1並行プログラムをソースプログラムとして記述する。このソースプログラムには、バグが潜在的に存在する可能性がある。

【0239】(2) ステップA2：逐次化  
デフォルト逐次性を導入することによって、第1並行プログラムを逐次構造の超逐次プログラムに変換する。本実施例では、メタレベルにおいてデフォルト逐次性を導入する。ここで、メタレベルとはソースプログラムそのもののレベルについていうのではなく、ソースプログラムの実行を管理するレベルをいう。例えば、並行プログラムで記述された第1並行プログラムを、それとは別に管理されるスケジューラによって逐次的に実行することをする。すなわち、ここでいう逐次化とは、CPファイル記憶部11と逐次化ルール記憶部13を対応付けることである。よって、得られる超逐次プログラムは、第1並行プログラムと逐次化ルールの対からなるとする。

【0240】(3) ステップA3：超逐次プログラムのテスト・デバッグ

ステップA2で逐次化された超逐次プログラムを実行し、テストを行う。この際のテスト実行は、第1並行プログラムを逐次化ルールに基づいて動作させることをいう。機能面のバグがある場合は、第1並行プログラムを修正し、ステップA2へ戻る。タイミング面のバグがある場合は、ステップJ1に移行する。タイミング面のバグがない場合は、その逐次化ルールを保存してステップA4に移行する。

【0241】(4) ステップJ1：逐次化ルールの修正

ステップA3でタイミング面のバグがある場合は、逐次化ルールを修正した後、ステップA2に戻る。

【0242】(5) ステップA4：良い非決定性の導入

良い非決定性の追加が必要な場合は、逐次化ルールを修正した後、ステップA2に戻る。良い非決定性の追加が必要ない場合は、ステップA7に移行する。

【0243】(6) ステップA7：並行化コンパイル  
ステップA3で保存された逐次化ルールから無害な非決定性部分を抽出し、その部分を並行化することで、部分超逐次プログラム全体を第2並行プログラムに変換する。

【0244】図82は、図79に示したようなプロセス間通信のある並行プログラムをメタレベルにおいてデフ

ォルト逐次性を導入して実行した結果の実行履歴であるログ情報の一例を示す図である。

【0245】ユーザは、このログ情報を出力装置5によって見る事ができる。図82において、縦のスクロールバーは任意の時刻のログ情報を見るために表示情報をスクロールするために用い、横のスクロールバーは良い非決定性を導入し複数の実行可能な候補がある時に、それら情報を見るために用いる。このように縦と横のスクロールバーを用いることにより、全ログ情報中の任意の場所を見る事ができる。更に、画面上部に示された「順序修正」と「非決定性導入」は、後述するそれぞれの操作を要求する時に、選択して処理を指示するために用いる。

【0246】更に、ユーザは入力装置4を介して指示を行うことによって、HSPファイル記憶部14に保存されている第1並行プログラムと実行ログ情報ファイルをテスト実行装置15に入力し、超逐次プログラムHSPとしてテスト実行を行うことができる。

【0247】テスト実行装置15は、テスト実行の結果を出力装置5に提示する。ユーザは、このテスト実行の結果に基づいて、超逐次プログラムHSPに対し入力装置4を用いて所定のテスト・デバッグを行うことができる。テスト・デバッグは、機能面の不具合についてはCPファイル記憶部11内の第1並行プログラムを修正し、タイミング面の不具合については逐次化ルール記憶部13内の実行ログ情報ファイルを修正し、再度、修正された第1並行プログラムと実行ログ情報を対応付けし直し、超逐次プログラムを再作成する。

【0248】ユーザは、超逐次プログラムHSPに対して前記テスト・デバッグを行った後、再度入力装置4からテスト実行の指示を与える。これにより、テスト・デバッグの行われた超逐次プログラムHSPは、テスト実行装置15に入力され、再度テスト実行が行われる。このテスト・デバッグは、超逐次プログラムHSPが正常に動作することを確認するまで繰り返し行われる。

【0249】テスト・デバッグにより正常に動作することを確認した時点で、逐次化ルール修正装置26により超逐次プログラムに対して良い非決定性に関する情報を部分的に導入していく。逐次化ルール修正装置26より導入された良い非決定性に関する情報は、ログ情報に反映され、HSPファイル記憶部14に記録される。

【0250】次に、逐次化ルール修正装置26により良い非決定性に関する情報が導入された超逐次プログラム（部分超逐次プログラムPHSP）について、ユーザからの指示に従ってテスト実行装置15によりテスト実行を行い、テスト・デバッグを行う。この場合、部分超逐次プログラムPHSPの振る舞いは、良い非決定性に関する情報の導入された部分については、非決定的な振る舞いをするので、その振る舞い全てについてテスト・デバッグを行うことが好ましい。このようにして、テスト

・デバッグ及び良い非決定性に関する情報の導入を繰り返し、良い非決定性に関する情報を徐々に付加していく。

【0251】良い非決定性に関する情報がインクリメンタルに導入され得られた部分超逐次プログラムPHSPは、ユーザからの指示により並行化装置18に入力される。この並行化装置18では、部分超逐次プログラムPHSPのうち、無害な非決定性部分を抽出し、部分超逐次プログラムPHSP全体を並行化する。即ち、集中ログ情報を各プロセス毎にその起動順序を規定するように分割し、各プロセス毎に起動制御を行うようにすることで、デフォルト逐次性を解除し、部分超逐次プログラムを並行プログラムに変換して第2CPファイル記憶部19に記録する。ユーザは、この並行プログラムを出力装置によって見ることができるとともに、最終的なテスト・デバッグを行うことができる。

【0252】図83は、まず、ステップA2～ステップA4の処理に関する部分超逐次プログラムを構成するまでの部分の構成例を示す。以下に、それぞれの構成部分の機能実現法について説明する。

【0253】図84に、プロセス間で交換されるメッセージのフォーマットを示す。「宛先情報」には宛先プロセス名等の情報伝達先を指定する情報がセットされ、「送信元情報」には送信元プロセス名等の送信元を規定する情報がセットされる。「メッセージ本体」には、プロセス間で交換するデータがセットされる。

【0254】ログ情報として保存される情報は、図84のメッセージフォーマット中の宛先情報と送信元情報を用いて生成される。以下の説明においては、ログ情報として宛先情報として宛先プロセス名を、送信元情報として送信元プロセス名だけを用いるが、同一プロセスが複数回利用される場合は、何回目の使用かの判別がつかなくなるので、送信元情報を「プロセス名とプロセス起動回数」からなるように拡張することができる。更に、あるプロセスの1回の起動中に複数回のメッセージ送信がある場合は、今回の起動後の何回目の送信かを特定するため、送信元情報を「プロセス名、プロセス起動回数、メッセージ送信回数」からなるように拡張することができる。更に、各プロセスが複数の処理の単位（メソッドと以降呼ぶ）を持つ場合には、その情報も付加して送信元情報は「プロセス名、メソッド名、メソッド起動回数、メッセージ送信回数」とする事もできる。

【0255】また、宛先情報も宛先プロセス名だけでなく、宛先メソッド名を付加して「宛先プロセス名とメソッド名」とすることもできる。もちろん、実行形態に応じて宛先情報と送信元情報を組み合わせることができるため、計8通りの組み合わせが存在する。

【0256】まず、実行装置22について詳述する。前述したように、第1並行プログラムをランダムに実行する。その際の実行ログを逐次化ルールとするため、本実

施例では、逐次化ルールとしてプロセス間の全ての通信を一旦プロセス群制御部203に送り、処理待ちメッセージ保存部204に蓄える。プロセス群制御部203は、処理待ちメッセージ保存部204から順次メッセージを取り出し、本来の宛先に対して再度送る。よってプロセス群制御部203によって全プロセスの起動が逐次的に制御されるため、第1並行プログラムは逐次的に処理されるようになる。

【0257】この時、プロセス群制御部203によって逐次的に処理されたメッセージは、ログ情報取得部202によって時系列にログ情報としてログ情報記憶部201に保存される。このログ情報が逐次化ルール記憶部13となり、以降の逐次実行時の実行制約条件となる。

【0258】図85は、プロセス間の通信をプロセス群制御部203を経由して行う様子を示す。この例では、プロセスAとBがプロセスCにメッセージを送信する時、プロセスAとBのどちらのメッセージが先にプロセスCに届いても良いとする良い非決定性が存在する場合であるが、実行時にたまたまプロセスAがプロセスBより先にメッセージを送信したとすると、プロセス群制御部203において「プロセスA→プロセスC」が先にログ情報に記憶され、続いて「プロセスB→プロセスC」が記憶される。図86に、その場合の逐次化ルールであるログ情報の一例を示す。

【0259】次に、逐次化装置12において、CPファイル記憶部11に記憶されている第1並行プログラムと上記のようにして得られた逐次化ルールであるログ情報を対応付ける。これにより、並行性を持った第1並行プログラムは、逐次化情報が付加された超逐次プログラムに変換される。

【0260】次に、テスト実行装置15におけるテスト・デバッグの段階では、プロセス間で交換されるメッセージをプロセス群制御部203において一旦受信し、ログ情報記憶部201に記憶されたログ情報の順序に従って本来の宛先に対して送信するようにする。このためプロセス群制御部203は、処理待ちメッセージ保存部204に保存されたメッセージ中から、ログ情報に従った順序でメッセージを取り出して送信する機構を有している。

【0261】上記のように、プロセス群制御部203でメッセージ送信順序をログ情報に指定された順序で送信するようにすることにより、並行プログラムの難しさの一つである処理順序の非決定性を解決することができ、処理の再現性が実現される。つまり、図85の例において、あるテスト実行時にはプロセスBの方がプロセスAより先にメッセージをプロセスCに送ったとしても、プロセス群制御部203でログ情報に図86に示すようにプロセスAからのメッセージを先にプロセスCに送る旨が記載されている場合は、プロセスBからのメッセージを処理待ちメッセージ保存部204に保存しておき、プ

プロセスAからのメッセージを受信しプロセスCに対して送信した後、処理待ちメッセージ保存部204に保存しておいたプロセスBからのメッセージをプロセスCに送る。

【0262】図87に、プロセス群制御部203の処理の流れを示す。

【0263】ログ情報記憶部201に保存されたログ情報に従って実行順番が一意に規定された超逐次プログラムにおけるテスト・デバッグは、機能面に不具合がある場合はソースプログラムを修正して上記処理ステップのステップA2から再度行う必要があるが、タイミングのみに不具合がある場合は、容易にテスト・デバッグをする事ができる。つまり逐次化ルールであるログ情報を修正し意図する順番にログ情報を書き換えれば良い。この書換は、ログ情報修正部205によって行う。逐次化ルールであるログ情報の修正は、ユーザからの指示により逐次化ルール修正装置26内のログ情報修正部205において行う。ログ情報修正部205は、図82に示すようにその処理した順番（時系列）にログ情報を表示することができる表示部を持つ。そしてタイミングのバグを修正したいユーザは、ログ情報の順序を入れ替える入替部を起動し、図88に示すように処理順序を交換したいメッセージを指定する。図88の例では、2メッセージ間の順序入れ替えを指示したが、複数メッセージの順序を入れ替えることも可能である。もちろん1メッセージ以上のメッセージを1つの集合としてとらえ、集合間での交換も可能である。そして、ログ情報修正部205には、ログ情報記憶部201のログ情報を入替部によって指定されたように書き換える書換部があり、図88の例の場合は図89のように変更される。また入れ替えメッセージの指定法としては、ポインティングデバイスを使って該当メッセージを指定する方法や、画面上の行番号をキーボードから指定する方法等がある。

【0264】上記のように、ログ情報を意図する順序に並べ替えることによって、テスト実行時にプロセス群制御部203が受信メッセージを本来の宛先プロセスに送信する順番を変えることになるため、タイミングの不具合を容易に修正することができるようになる。

【0265】例えば、図85において本来プロセスCはプロセスBからのメッセージをプロセスAからのメッセージより先に処理しなければならない場合に、ログ情報に図86に示すようにプロセスAからのメッセージを先に処理するようになっていたならば、上記の手順に従ってログ情報を図89に示すように修正すればよい。これによって、プロセス群制御部203のメッセージ処理順番が変更されるため、処理のタイミングの不具合を解決することができる。

【0266】このようなタイミングの不具合解決方法を用いることによって、プロセスの処理順序を変更するためにソースプログラムを修正することなく、容易にテス

ト・デバッグをすることが可能となり、プログラム開発の生産性が向上する。

【0267】次に、良い非決定性の導入方法について説明する。

【0268】良い非決定性は、ログ情報修正部205内の非決定性導入部によってログ情報を修正することで行う。これは、超逐次化されたプログラムにユーザが意図的に非決定性を導入するものである。例えば図85に示す処理において、プロセスAとプロセスBのどちらが先にプロセスCにメッセージを送っても良い場合には、対象となるログ情報を図90に示すように指定することによって行う。これによって、非決定性導入部は、指定されたメッセージの順序制約を解除した形で、例えば図91(a)、図91(b)に示すようにログ情報記憶部201に記憶されたログ情報を修正する。図91(b)は、図91(a)を横スクロールバーで右ヘシフトするように指示することによって得られる。この場合は2つのメッセージ順序の非決定性を導入した場合であるが、更に多数のメッセージ順序の非決定性を導入する場合は、順次右ヘシフトすることによってその一覧を見ることが出来る。

【0269】図91では、プロセスAないしプロセスBからのメッセージであれば、プロセスCにメッセージを送り、続いて次のタイミングでも、プロセスAないしプロセスBからのメッセージであれば、プロセスCにメッセージを送るように解釈する。プロセス群制御部203が上記のようにログ情報を解釈するようにすることによって良い非決定性を扱うことが可能となる。これは、プロセス群制御部203の処理における図87におけるステップK3において、メッセージ送信元として一致するか否かのチェック対象が、一つから複数個に拡張されることによって実現できる。

【0270】但し、上記例では、プロセスCがプロセスAないしプロセスBから続けて2つのメッセージを受信した場合でも、それらのメッセージを処理してしまう問題点があるが、これは、先に受信したメッセージを次の起動候補から削除することによって容易に防止することができる。削除対象となるメッセージは、本実施例では説明の容易のために、送信先情報をプロセス名のみで示したが、図84の送信先情報説明で前述したように唯一に特定できるので、未処理メッセージの情報を削除することはない。

【0271】よって、この良い非決定性の導入によって、外部からの非決定的な刺激に適切に反応でき、プロセスの柔軟性、再利用性、拡張性を実現できる。

【0272】上記の実施例では、良い非決定性を2つのメッセージの到着順序に関して示したが、2つ以上の複数のメッセージの到着順序に良い非決定性を与えることも可能である。その場合でも、上記処理と同様にプロセス群制御部203の処理における図87のステップK3

においてメッセージ送信元として一致するか否かのチェック対象を、一つから複数個に拡張することによって解決されるので、正しく動作することは明らかである。

【0273】次に図81におけるステップA7と図80の並行化装置18に関する並行化コンパイルのフェーズについて詳述する。図92に、ステップA7の処理に関する部分の構成例を示す。ここで、集中ログ情報記憶部301はHSPファイル記憶部14内に保存されている良い非決定性導入後の実行ログ情報を保存している記憶部である。例えば図79におけるプロセス群が図93に示すようなメッセージ交換を行った場合、図82で示したような超逐次プログラム用の集中ログ情報から、例えば図94のように良い非決定性が導入された集中ログ情報に変換されたものが保存されているとする。

【0274】以下には、良い非決定性が導入された集中ログ情報を各プロセス毎に分割する手順を図92、図94、図95を用いて説明する。集中ログ情報を分割する基準は、分割基準指定装置23内の分割基準指定部303から指定する。この基準としては、例えば宛先プロセス名であったり、宛先計算機番号であったりするが、以下の説明では、宛先プロセス名を用いる。宛先名にメソッド名も付加する場合には、分割基準として、〔宛先プロセス名、メソッド名〕を指定して、宛先メソッド毎に分割して分割ログを作ることも可能である。分割基準指定部303では、この分割基準を保持している。

【0275】ユーザからの指示により集中ログ情報を分割させようとする、ログ情報分割部302が起動される。ログ情報分割部302は、図95に示す手順に従って集中ログ情報を分割基準指定部303によって指定された基準で分割する。

【0276】ログ情報分割部302が起動されると、分割基準指定部303より分割基準値を読み込み（ステップL1）、分割基準を把握する。続いて集中ログ情報記憶部301に保存されているログ情報を1レコード読み込み（ステップL2）、最終レコードか否かチェックする（ステップL3）。この例では、集中ログ情報の1レコード目は、図94に示すようにプロセスAからプロセスDへのメッセージなので、最終レコードでないと判断し、分割基準（宛先プロセス名=プロセスD）に従って、該当する分割ログ情報記憶部（ここでは、分割ログ情報記憶部Dとする）に保存する（ステップL4）。この操作を集中ログ情報記憶部301に記憶されているログ情報が終了するまで繰り返す。この結果、図96に示すように宛先プロセス毎に分割されたログ情報が分割ログ情報記憶部A、B、C、Dのそれぞれに保存される。

【0277】図96（a）では、例えばプロセスAはプロセスCからのメッセージを処理した後に、プロセスDからのメッセージを処理することを規定している。また、図96（c）のプロセスCは、いわゆる良い非決定

性を導入した処理であり、プロセスAないしプロセスBからのメッセージを最初に処理し、次もプロセスAないしプロセスBからのメッセージを処理することを規定している。これは言い替えると、プロセスAからのメッセージとプロセスBからのメッセージのどちらが先に来ても良いという良い非決定性を表現している。ここでは、2つのメッセージの順序の制約を解除した例を示したが、より多くのメッセージの順序制約を解除し、良い非決定性を導入したい場合は、同一行に所望数のメッセージを列挙すれば良い。

【0278】また、分割ログ情報は宛先プロセス毎に分割されるため、分割ログ情報中の送信先プロセス名を省略することもできる。

【0279】次に、このようにプロセス毎に分割されたログ情報に従って各プロセスが動作する方式の一例を述べる。各プロセスは、例えば図97に示すようにメッセージを受信し実際のプロセスの処理を開始する前にプロセス制御部305を呼出し、図98に示す処理を行う。このようなプロセス制御部の呼出は、並行化装置18の一部として第1並行プログラムにおける各プロセスの処理の先頭に呼出処理を挿入するようにソースプログラムを書き換えることによって行うことができる。即ち、各プロセスは図99に示すように、当該プロセス本来の処理部分に加え、プロセス制御部305、分割ログ情報記憶部304、処理待ちメッセージ保存部306からなる。プロセス制御部305は、当該プロセスに内在する分割ログ情報記憶部304に保存されているログ情報を参照してメッセージ処理を行い、処理すべきメッセージでない場合は、処理待ちメッセージ保存部306に保存する。

【0280】次に、プロセス制御部305の処理について図96と図98を用いて説明する。

【0281】例えばプロセスAは、図93に示す処理の流れの場合、プロセスCとプロセスDのどちらから先にメッセージをもらうかは決定されておらず、ここに処理タイミングの非決定性が存在している。しかし、ここでは超逐次で実行したときの実行ログ（図82相当）より、プロセスCからのメッセージを先に処理し、その後にプロセスDからのメッセージを処理するように規定される。よって、無害の非決定性を導入したとしても、このメッセージ処理順序の規定は守られなければならない。

【0282】最初の例として、プロセスAが最初にプロセスCからのメッセージを受信して起動された場合を示す。プロセスAがプロセスCからのメッセージを受信して起動されたならば、プロセス制御部が呼び出され、当該プロセスを起動させたメッセージの送信元情報を得る（ステップM1）。ここでは、送信元情報 $!s = \{\text{プロセスC}\}$ となる。続いて分割ログ情報記憶部よりまず最初に来るべきメッセージ情報 $!r1$ を得る（ステップM

2)。ここでは、 $I r 1 = \{\text{プロセスC}\}$ となる。

【0283】次に、現在受信したメッセージが分割ログ情報に記憶された順序に一致しているかを調べるため、 $I s$ が $I r 1$ に含まれているかどうかを調べる(ステップM3)。この場合は、ともにプロセスCで一致含まれているため、受信メッセージに基づいた処理を即行うことが許される。よって次の起動チェックのために分割ログ情報記憶部から読み出すレコードを1つ進めておく(ステップM4)。そして、プロセス本体の処理を開始する前に、処理待ちメッセージ保存部に実行待ちメッセージが存在している場合には、その保存されたメッセージが起動可能となったかどうかを調べる。つまり、分割ログ情報記憶部に保存されている情報から次に処理すべきメッセージ情報 $I r 2$ を取り出し(ステップM5)、それに含まれるメッセージが処理待ちメッセージ保存部に存在しているか調べる(ステップM6)。この場合は、 $I r 2 = \{\text{プロセスD}\}$ となるが、処理待ちメッセージ保存部にメッセージが存在しないため処理を終了し、プロセスCからのメッセージに基づいた本処理を行う。

【0284】続いて、プロセスA宛にプロセスDからのメッセージが届いたならば、上記同様に処理を行うと、 $I s = \{\text{プロセスD}\}$ 、 $I r 1 = \{\text{プロセスD}\}$ となり、集合 $I s$ は集合 $I r 1$ と等しいか又は $I r 1$ に含まれる。 $I r 2$ は存在しないため、処理待ちメッセージ保存部内のメッセージを調べる必要なく処理を終了し、プロセスDからのメッセージに基づいた処理を行う。

【0285】また、逆にプロセスAが、プロセスDから先にメッセージを受信した場合の処理の流れを示す。つまりプロセスAが、最初にプロセスDからのメッセージを受信して起動されたならば、プロセス制御部が呼び出され、当該プロセスを起動させたメッセージの送信元情報を得る(ステップM1)。ここでは、送信元情報 $I s = \{\text{プロセスD}\}$ となる。続いて、分割ログ情報記憶部よりまず最初に来るべきメッセージ情報 $I r 1$ を得る(ステップM2)。ここでは $I r 1 = \{\text{プロセスC}\}$ となる。

【0286】次に、現在受信したメッセージが分割ログ情報に記憶された順序に一致しているかどうかを調べるため、 $I s$ が $I r 1$ に含まれているか調べる(ステップM3)。この場合は、含まれないため即プロセスDからのメッセージを処理してはいけないと判断し、処理待ちメッセージ保存部306にプロセスDからのメッセージを保存し(ステップM8)、プロセスAの処理を強制的に終了させる(ステップM9)。即ち、プロセスDからのメッセージに基づいた処理を行うことなくプロセスAの処理を終了する。

【0287】続いて、プロセスA宛にプロセスCからのメッセージが届いたならば、上記同様に処理を行うと、 $I s = \{\text{プロセスC}\}$ 、 $I r 1 = \{\text{プロセスC}\}$ となり

$I s$ は $I r 1$ に含まれるため、受信メッセージに基づいた処理を即行うことが許される。よって次の起動チェックのために分割ログ情報記憶部から読み出すレコードを1つ進める(ステップM4)。そして、プロセス本体の処理を開始する前に、処理待ちメッセージ保存部に起動可能となった実行待ちメッセージが存在しているかどうかを調べる。つまり、分割ログ情報記憶部に保存されている情報から次に処理すべきメッセージ情報 $I r 2$ を取り出し(ステップM5)、それに含まれるメッセージが処理待ちメッセージ保存部に存在しているか調べる(ステップM6)。この場合は、 $I r 2 = \{\text{プロセスD}\}$ となり、処理待ちメッセージ保存部に先ほど保存しておいた該当メッセージが存在するため、その実行可能となったメッセージを次に処理するために、送信先(ここではプロセスA)宛に送信元情報等をそのまま(この例では、プロセスDのまま)送信する(ステップM7)。そして、プロセスAは受信メッセージとしてプロセスCから送られたメッセージに基づく処理を行うべくプロセス制御部の処理を終了する。

【0288】更に、プロセスAは前記処理待ちメッセージ保存部から取り出し送信したメッセージを受信するので、上記同様に処理を行うと、 $I s = \{\text{プロセスD}\}$ 、 $I r 1 = \{\text{プロセスD}\}$ となり $I s$ は $I r 1$ となる。 $I r 2$ はもはや存在しないため、処理待ちメッセージ保存部内のメッセージを調べる必要なく処理を終了し、プロセスDからのメッセージに基づいた処理を行う。

【0289】このことより、たとえメッセージの到着順序が実行ログに記載された順序と異なっていたとしても、実行ログに記載された順序に処理され、非決定的な処理の再現性が保証される。

【0290】次に良い非決定性を導入した場合の処理例として、プロセスCの動作を例に示す。プロセスCが最初にプロセスAからのメッセージを受信して起動されたならば、プロセス制御部が呼び出され、当該プロセスを起動させたメッセージの送信元情報を得る(ステップM1)。ここでは、送信元情報 $I s = \{\text{プロセスA}\}$ となる。続いて分割ログ情報記憶部よりまず最初に来るべきメッセージ情報 $I r 1$ を得る(ステップM2)。ここでは $I r 1 = \{\text{プロセスA、プロセスB}\}$ となり、プロセスAからのメッセージでもプロセスBからのメッセージでも、どちらからのメッセージでも良いとしている。

【0291】次に、現在受信したメッセージが分割ログ情報に記憶された順序に一致しているかを調べるため、 $I s$ が $I r 1$ に含まれているかどうかを調べる(ステップM3)。この場合、集合 $I s$ は集合 $I r 1$ と等しいか又は $I r 1$ に含まれるため、受信メッセージに基づいた処理を即行うことが許される。よって、次の起動チェックのために分割ログ情報記憶部から読み出すレコードを1つ進めておく(ステップM4)。そして、プロセス本体の処理を開始する前に、処理待ちメッセージ保存部に

実行待ちメッセージが存在している場合には、その保存されたメッセージが起動可能となったかを調べる。つまり、分割ログ情報記憶部に保存されている情報から次に処理すべきメッセージ情報  $Ir2$  を取り出し（ステップM5）、それに含まれるメッセージが処理待ちメッセージ保存部に存在しているか調べる（ステップM6）。この場合は、 $Ir2 = \{\text{プロセスA、プロセスB}\}$  となるが、処理待ちメッセージ保存部にメッセージが存在しないため、処理を終了し、プロセスAからのメッセージに基づいた本処理を行う。

【0292】続いて、プロセスC宛にプロセスBからのメッセージが届いたならば、上記同様に処理を行うと、 $Is = \{\text{プロセスB}\}$ 、 $Ir1 = \{\text{プロセスA、プロセスB}\}$  となり、 $Is$  は  $Ir1$  と等しいか又は  $Ir1$  に含まれる。 $Ir2$  は存在しないため、処理待ちメッセージ保存部内のメッセージを調べる必要なく処理を終了し、プロセスBからのメッセージに基づいた処理を行う。

【0293】上記の良い非決定性導入の例では、メッセージの到着順序が逆の状態、即ちプロセスCに、プロセスBからのメッセージが先に到着し、その後プロセスAからのメッセージが到着した場合も同様に処理できることは明かである。

【0294】但し、上記例では、プロセスCがプロセスAないしプロセスBから続けて2つのメッセージを受信した場合でも、それらのメッセージを処理してしまう問題点があるが、これは、先に受信したメッセージを次の起動候補から削除することによって容易に防止することができる。削除対象となるメッセージは、本実施例では説明の容易のために、送信先情報をプロセス名のみで示したが、図84の送信先情報説明で前述したように唯一に特定できるので、未処理メッセージの情報を削除することはない。

【0295】上記のように、オブジェクト毎に実行ログ情報を分割して動作させることによって、並行プログラムが潜在的に持っている無害な非決定性を実現しながら、超逐次プログラムで抽出した逐次性を維持しているため、処理の再現性を保持しており、超逐次プログラム実行時と同一結果を、より高い処理性能で実現することが可能となる。

【0296】また、上記の例では良い非決定性を2つのメッセージの到着順序に関して示したが、2つ以上の複数のメッセージの到着順序に良い非決定性を与えることも可能である。その場合でも、上記処理は  $Ir1$  ないし  $Ir2$  が複数の要素を持つことによって解決されるので、正しく動作することは明かである。

【0297】また、上記例では、各プロセスないし各モジュールの処理の始めにプロセス制御部を呼び出して実現する実施例を示したが、各計算機の実オペレーティングシステムが、宛先プロセス毎の処理待ちメッセージ保存部を保持し、メッセージ処理の都度、該当する分割ログ情

報を参照して、該当プロセスを起動すること又は処理待ちメッセージ保存部に保存することによって実現することも可能である。

【0298】この場合は、図98におけるステップM7で次の処理対象メッセージを送信することなく、実行待ち状態から次に起動可能なプロセスとしてスケジューリング対象に変更すれば良い。

【0299】本実施例によれば、ソースプログラムを修正することなく逐次化ルールであるログ情報を書き換えることにより、処理タイミングの非決定性に起因する不具合を容易に解決することができる。このことによって、処理タイミングの非決定性が内在する並行/並列/分散プログラムの開発を容易にし、生産性を向上させることができる。

【0300】また、本実施例では、ユーザの意図する良い非決定性のみを容易に導入することが可能になるため、並行プログラムとしての柔軟性、再利用性、拡張性を維持することもできる。

【0301】更に、並行プログラムを逐次化して得られた全プロセスの集中ログ情報を各プロセス毎に分割し、分割されたログ情報に基づいて各プロセスを制御することによって、無害の非決定性を自然に導入し、集中ログ情報に基づく超逐次プログラム実行時と同一結果を高い処理効率で得ることができる。

【0302】尚、本実施例に係るコンピュータシステムの構成は、図1において、(a) 共有メモリ3が存在しない構成の場合、(b)  $I/O$ インタフェース2がバスで密にプロセッサ1-1~1-Nが接続している並列計算機の場合、(c)  $I/O$ インタフェースが通信路で疎に結合している分散ネットワーク計算機構成の場合、(d) 単一プロセッサで構成される場合、及びそれらの組み合わせの場合、が考えられる。

【0303】（実施例9）本実施例で使用される新しい概念であるセクションについて説明をする。

【0304】本実施例におけるセクションとはスレッドの入口を1個のみ持ち、出口を複数持つプログラム断片を言う。ここでスレッドとは、コンピュータ言語の分野で一般的に用いられる用語であり、プログラムをたどりながら実行をする作用体をいう。例えばスレッドが同時に複数走った場合には、それは並行実行を意味している。

【0305】下記のようなプログラム断片があったとする。

```
code 1
code 2
:
branch 処理1
処理2
```

ここで、簡単のため `code 1` から `branch` 命令までの区間で、区間外への分岐命令又は同期命令はないも

のとする。このプログラム断片をスレッドが処理していくとき、code 1が実行の開始点、すなわち、code 1がスレッドの入口となる。次に、処理が進みbranch命令に到達した時、branch命令までの処理内容に応じて処理1に分岐するか処理2に進むかが決まるとすれば、スレッドが処理1に進む場合と処理2に進む場合の2個の出口があることになる。従って、code 1からbranch命令までのプログラム断片はスレッドの出口を2個持つセクションとなる。ところで、もしcode 1からbranchまでの間でこのプログラム断片から外部への分岐命令があった場合は、そここのセクションの出口の一つとなる。

【0306】上記の例は、1本のスレッドが複数の出口から1個の出口を選択して出ていく場合を示したが、次にプログラム断片の内部でスレッドの生成が行なわれる場合の例を示す。

```
code 1
code 2
:
fork
```

処理1

上記の例では、fork命令でスレッドの分裂、すなわちもう一つのスレッドの生成が行なわれる。この時、処理1は分裂した2本のスレッドによって並行に処理されるが、これを2個の出口から各々のスレッドが出て行き、処理1を実行するものとみなすことができる。従って、code 1からfork命令の間は、この区間に外部への分岐命令がなければ、出口を2個持つセクションとなる。このように、プログラム断片の内部でスレッドの生成が行なわれる場所も一種の分岐とみなされ、複数の出口をもつセクションとなり得る。

【0307】図100は、上記のようなセクションを概念的に表した図である。図100に示すように、セクションは、スレッドの入口101と、セクションの本体部分102と、スレッドの複数の出口103とからなる。以後セクションを概念的に図示する時にはこのような表現を用いる。

【0308】次にセクションの融合と分割を説明する。図101はセクションの融合を表す概念図である。図101は、S11とS12の2個のセクションが融合してS13という新たなセクションを生成する例を示している。図101中の矢印201はS11の出口の一つであり、それをS12の入口に接続することで2個のセクションの融合を可能としている。この例では2個のセクションの融合を示したが、一般に複数のセクションにおいて出口と入口を接続することにより、新たなセクションを生成（融合）することができる。

【0309】図102はセクションの分割を表す概念図である。図102は、セクションS21を2個のセクションS22とS23に分割する例を示している。こ

で、以下のプログラム断片の例を考える。

```
code 1
:
code 2
code 3
:
code 4
```

ただし、code 1～4には分岐命令を含まないものとする。これをスレッドが処理する時、code 1が入口でcode 4が出口となることから、このプログラム断片を出口が1個のセクションとみなすことができる。このプログラム断片をセクションS21に対応させる。次に、例えば、code 2とcode 3の間で分割して、code 1～code 2をセクションS22、code 3～code 4をセクションS23に対応させれば、セクションS21はセクションS22とセクションS23に分割されたことになる。このように1個のセクションを複数のセクションに分割することが可能である。

【0310】次にセクションの分類について説明する。セクションは基本セクションとグループセクションに分類される。基本セクションは、内部に分岐命令や同期命令を含まないセクションをいう。基本セクションを分割した場合には、基本セクションになる。グループセクションは基本セクション以外のセクションをいい、グループセクションは複数の基本セクションに分割することができる。

【0311】セクションに予測実行時間を付加する場合を説明する。ここではセクションを構成する個々の命令の実行時間（又は平均実行時間）がわかっているものとし、この時間を各命令の予測実行時間と呼ぶことにする。

【0312】基本セクションは途中で分岐することなく直列に実行されるので、セクションを構成する各々の命令の予測実行時間を合計することにより、そのセクションの入口から出口までの予測実行時間が計算される。基本セクションの場合は、出口が複数あってもセクションの出口は必ず命令列の最後の分岐命令又は同期命令であるため、一般に入口から各出口までの予測実行時間は等しくなる。

【0313】グループセクションは基本セクションに分解できるため、グループセクションの入口から各出口までに通過する基本セクションの予測実行時間を加えることにより、出口までの予測実行時間が計算される。すなわちグループセクションでは出口によって予測実行時間は変わってくる。また、グループセクション内部に処理結果に応じて回数が変わるループや無限ループ、join等の同期命令、入出力命令を含む実行時間の予測は不可能である。その時は、グループセクションを、上記のような実行時間の予測ができないようなループ、同期命令、入出力命令を内部に含まない複数のセクションに分



割し、分割されたセクションを対象とすれば良い。

【0314】上記の方法によりセクションの各出口までの予測実行時間が計算された後に、その中で最大のものをセクション自身の予測実行時間とする。

【0315】セクションの予測実行時間は、セクションの実行の効率化のためにセクションの融合・分割をするときの指針となり、プログラム自身を最適化するときの指針としても利用できる。

【0316】本実施例では同期の容易さ、実行時間の予測の容易さのため、fork等のスレッドを生成する命令とjoin等の同期命令、入出力命令はセクションの内部には含まれずセクションの終端に位置し、実行時間の予測不可能なループは内部に持たないものとする。

【0317】実施例9を図面を参照して具体的に説明する。図103は、本発明の実施例9の概略構成を示す図である。

【0318】プログラム解析部411は、図示されない入力部から入力されたソースプログラムを解析し、基本セクションへの分割及びセクションの実行順序に関するルールの抽出をおこなう。プログラム解析部411で解析された情報（すなわちセクション情報）はセクション情報記憶部412に記憶される。

【0319】セクション情報編集部413はセクション情報記憶部412に記憶されたセクション情報の編集を行なう。具体的には、セクションの実行順序に関するルールの編集やセクションの融合・分割が、自動的に又はユーザによって行なわれる。コンパイル部414は各セクションをコンパイルしてオブジェクトコードに変換し、その結果をセクション情報記憶部412に登録する。

【0320】実行部415はセクション情報記憶部412に記憶されたセクション情報に基づき、セクションを単位として逐次又は並行にプログラムを実行していく。実行方法の逐次と並行の切替えは、逐次実行モードと並行実行モードの2種類のモード間を切替えることによって行なわれる。逐次実行モードで実行をすれば並行プログラムの再現性の問題を回避することができる。すなわち何度実行しても同じ結果が得られるため、並行プログラムについて有効なテスト・デバッグを行なうことができる。その状態で並行実行モードに切替えて実行すれば、セクション単位の同期をとりながら実行をすることにより、安全な並行実行を行なうことができる。なお、実行部415がインタプリタ的にのみプログラムを実行する場合は、コンパイル部414は省略することができる。

【0321】実施例9を構成する各部分について詳細に説明する。

【0322】まず、図103に示されるセクション情報記憶部412について、特にセクションに関する情報がどのような構造で記憶されるのかを詳細に説明する。セ

クション情報は、セクション自身に関する情報とセクションの実行順序に関する情報（ルール）とに2分される。まずセクション自身がどのように記憶されているかを図104を用いて説明する。

【0323】図104は、出口が3個のグループセクション2個と出口が2個の基本セクション1個とから構成される例を示し、全体として出口が6個のグループセクションの記憶方法を示している。セクション情報記憶部412は、図104に示すような、セクションの管理テーブル501、505、506、509を有する。これらの管理テーブルには図105に示すように、セクションのラベルと、セクションがコンパイルされているかを示すフラグと、コンパイルされているならそのオブジェクトを指すポインタと、セクションがプログラムコードかを示すフラグと、このセクションを構成するセクションの部品テーブルを指すか又はプログラムコードを指すポインタと、各出口を指すポインタとその出口までの予想実行時間等が記憶される。

【0324】セクション情報記憶部412は、更に、図104に示すセクションの部品テーブル502、503、504を有する。部品テーブルとはグループセクションを構成する個々の部品としてのセクションについて記憶するテーブルである。具体的には、部品テーブルは、図106に示すように、部品としてのセクションの管理テーブルを指すポインタと、各出口がどのセクションに接続されているかを指すポインタ等が記憶される。図104に示す例では部品テーブル502は他のグループセクションの管理テーブル505を指すポインタと、3個のうち2個の出口の接続先のセクションの、部品テーブル503、504を指すポインタを記憶している。ここで、例えば部品テーブルにおける出口508はどの部品テーブルにも接続されていない。この時、その出口を指すポインタは、管理テーブル501にセクションの出口として記憶される。また、管理テーブル506がプログラムコード507を指しているように、管理テーブルはプログラムコードも同様に管理している。

【0325】セクションの実行順序に関するルールを説明する。この場合において、スレッド1が以下のプログラム断片を実行するとする。

処理1

join スレッド2

処理2

上記のセクションは、スレッド1が処理1を終了させた後、スレッド2の終了を待ち、処理2を行なう、ということの意味している。図107は上記の例をセクションの概念図に置き換えて図示したものである。S31は上記の処理1を構成するセクションのうち最後のセクションである。S32は上記の処理2を構成するセクションのうち最初のセクションである。S33はスレッド2が実行する最後のセクションである。この時、このjoin

n文によって、

セクションS31の実行後、スレッド2におけるセクションS33の実行が終了してからセクションS32の実行が開始される

というセクション実行順序のルールが抽出され、セクション情報記憶部412に記憶される。このようにセクションの実行順序に関するルールは、主に実行時にセクション間の同期をとるために利用される。

【0326】図108は図103におけるプログラム解析部411の詳細な構成を示す。図108において中間言語変換部901は図示されない入力部から入力されたソースプログラムを中間言語に変換する。中間言語変換部901で中間言語に変換するのは、セクションの抽出がしやすくなることと、各命令の実行時間が予測しやすくなるという理由による。なお、中間言語変換部901での変換に当たっては従来のコンパイラの技術がそのまま使用できる。セクション情報抽出部902は、中間言語変換部901によって変換された中間言語から基本セクションを切り出し、セクションの実行順序に関するルール等を抽出する。

【0327】プログラム解析部411における処理の流れを図を用いて詳細に説明する。図109はプログラム解析部411における処理の流れを示した図である。

【0328】入力されたソースコードは、中間言語変換部901において中間言語に変換される(ステップN1)。変換された中間言語はセクション情報抽出部902に引き渡される。

【0329】セクション情報抽出部902は、中間言語から基本セクションの切り出しや実行順序のルールを抽出するために初期化される(ステップN2)。この初期化は新しいセクションの切り出しのたびに行なわれる。この例では、基本セクションを切り出しながら同時にそのセクションの予測実行時間を計算するため、それに関する初期化も行う。

【0330】中間言語を1行ずつ読み込み、処理を繰り返していく。この繰り返しの中で中間言語の読み込みがすべて終了した時は(ステップN3)、この時点まで読み込んだ中間言語列を1個の基本セクションとしてセクション情報記憶部412に記憶し(ステップN8)、処理を終了する(ステップN9)。

【0331】中間言語の読み込みが終了していないのであれば、更に中間言語を1行読み込み(ステップN4)、読み込んだ中間言語が、例えばjoinセクションの実行順序のルールを決める命令ならば(ステップN5)、それによって決定されるルールをセクション情報記憶部412に登録する(ステップN10)。中間言語が分岐又は同期命令ならば(ステップN6)、その命令は基本セクションの切れ目となるので、初期化以降に読み込んだプログラム断片を1個の基本セクションとしてセクション情報記憶部412に記憶し(ステップN11)、次の

セクションの抽出を行うために、ステップN2に戻る。ステップN4において、読み込んだ中間言語の命令が上記の条件を満たさなければ、その命令の予測実行時間を切り出し中のセクションの予測実行時間に加え(ステップN7)、ステップN3に戻る。ここで中間言語の各命令の予測実行時間は、あらかじめ測定をする等してテーブル等に記憶させておけば良い。以上の手続きが終了した時点で、もとのプログラムは、中間言語列として切り出された基本セクションと、セクション実行順序のルールとに変換され、セクション情報記憶部412に記憶されたことになる。

【0332】セクション情報編集部413の詳細な説明をする。セクション情報編集部413は大きく二つの動作をする。一つはセクションの融合・分割をする動作であり、もう一つの動作は、セクションの実行順序のルールを編集する動作である。

【0333】セクションの融合・分割の主な目的としては、セクションの大きさを揃えることによって、実行部415においてより効率的な実行を行なわせることにある。本実施例では、この動作にセクションの予測実行時間を利用している。セクション情報編集部413では、各セクションの予測実行時間がほぼ均一になるようにセクションの融合・分割をすれば良く、これは容易に自動化できる。

【0334】また、セクション情報編集部413のセクションの実行順序のルールの編集により、実行時におけるセクション間の同期や、実行順序等を自由に変更することができ、効果的なテスト・デバッグを可能とする。

【0335】図110を用いてセクション情報編集部413の詳細な構成を説明する。図110においてセクション情報変換部1101は、図示されない入力装置により入力されたセクションの融合・分割の指示、セクションの実行順序のルールの編集要求に従い、セクション情報記憶部412に記憶されたセクション情報を変更する。この動作は主にユーザによってなされるが、セクションの融合・分割に関してはセクションの予測実行時間を利用することにより容易に自動化をすることができる。セクション情報表示部1102はセクション情報記憶部412に記憶されたセクション情報を、セクション情報変更部1101からの指示に従い表示する。

【0336】図111は図103におけるコンパイル部414の詳細な構成を示す。図111において、セクション末端処理部1601は、セクションの末端に次に実行すべきセクションの登録とセクションの終了のためのコードを書き込む。セクションコンパイル部1602は、末端処理を受けたセクションをコンパイルしてオブジェクトに変換し、セクション情報記憶部412に登録する。

【0337】ここで、セクションの末端の処理の仕方を説明する。例えば、あるセクションの末端が以下のよう

なコードであったとする。

処理1

```
branch label
```

処理2

```
label:
```

処理3

上記のコードは、処理1の結果によって、labelにジャンプして処理3を実行するか、そのまま処理2を実行する、ということを表したコードである。ここで、処理2の先頭に位置するセクションをセクション2、処理3の先頭に位置するセクションをセクション3、としたとき、上記コードは以下のように処理される。

処理1

```
branch label
```

```
section-end セクション2
```

```
label:
```

図112は図103における実行部415の詳細な構成図を示す。図112において、実行セクション選択部1201は、セクション情報及び実行セクション候補記憶部1203に記憶される情報に基づき、次に実行すべきセクションを1個又は複数選択する。実行セクション選択部1201は、図示されない入力部から切替えることが可能な2つのモード（逐次実行モードと並行実行モード）を有する。逐次実行モードではセクションの選択は1個である。並行実行モードではセクションの選択は一般に複数となる。セクション実行部1202は、実行セクション選択部1201によって選択されたセクションを実行する。特に、選択されたセクションが複数の場合は並行に実行する。実行セクション候補記憶部1203は、セクション実行部1202で実行した結果から決められる次の実行の候補となるセクションを記憶する。

【0338】図113は、実行部415における処理の流れを示す図である。プログラムは、セクション情報記憶部412に記憶されるセクションの情報及び実行順序のルールに従って、セクション単位に実行される。

【0339】全体の初期化が行なわれる（ステップ1）。この時、実行セクション候補記憶部1203には最初に実行すべきセクションが記憶される。

【0340】次にセクション毎に実行を行なうループに入る。このループの最初では、次に実行するセクションの選択が、モードによって異なる方法で行なわれる（ステップ2）。逐次実行モードでは、実行セクション候補記憶部1203に記憶される次に実行されるセクションの候補の中から実行順序のルールに従うものを1個選択する（ステップ3）。一般に候補となるセクションの中には、ルールに従うものが複数ある。これから1個を選択する方法としては、例えば記憶された時間が最も前のものを選択しても良いし、又は、固定された乱数列に従って選択しても良い。ただし、ここでの選択方法としては再現性が保証される方法を用いる必要がある。プ

ログラムの実行のたびに選択される順序が変化しない

（すなわち再現性が保証されている）ことは本発明の重要な動機の一つだからである。モードが並行実行モードでは、実行候補の中からルールに従うセクションをすべて選択する（ステップ4）。選択されたセクションは一般に複数となる。

【0341】ステップ3又はステップ4において、選択に失敗した場合は（ステップ5）、プログラムが終了したか、実行順序を決めるルールの異常により失敗した場合が考えられる。そこで、候補のリストを確かめ（ステップ6）、候補が残っていないならば正常終了（ステップ7）、残っているならば異常終了（ステップ8）とする。異常終了の場合には、例えばデッドロック状態が考えられ、この旨をユーザに提示することによりテスト・デバッグの指針とすることができる。ステップ3及びステップ4において、選択に成功した時は、選択されたセクションの実行をする（ステップ9）。

【0342】複数のセクションが選択された時には、それらのセクションは並行に実行される。この後セクション実行部415は、並行に実行されたセクションすべての実行の終了を待ってから、次のステップに進む。

【0343】なお、本実施例で使用しているセクションの内部には、無限ループ等の実行時間を予測不可能にする因子は含まれていないため、セクションの実行終了を永久に待ち続けるということはない。

【0344】実行が終了すると、まず終了したセクションが実行セクション候補記憶部1203から消去され、次に実行結果に応じて次の実行候補となるセクションが記憶される（ステップ10）。「実行結果に応じて」ということは例えばセクションの最後がif文のときは、計算結果によって次に分岐する分岐先、すなわち次に実行すべきセクションが変わる、ということの意味する。また、セクションがend文で終了するときは、それを実行していたスレッドも終了するので、このスレッドが次に実行すべきセクションは無いことになる。この時は、実行セクション候補記憶部1203には何も記憶されない。

【0345】実行セクション候補記憶部1203での記憶内容の詳細な変化を説明する。図114は、プログラムの一例をセクションの概念図を使って表した図である。1本のスレッドがセクション1を実行し、その中のfork文により2本のスレッドに分裂し、それぞれのスレッドがセクション2とセクション4を実行し、join命令により同期をとって1本のスレッドに戻り、最後にセクション3を実行して終了する例を示す。ここでjoin文により同期をとるために、セクション3はセクション4の後に実行されるというセクションの実行順序に関するルールが、事前のプログラム解析により抽出されているとする。

【0346】図115は上記の例を、逐次実行モードで実行した時の実行セクション候補記憶部1203の変化を示す図である。

【0347】初期状態1401では最初の実行されるセクション1が記憶されている。そこで、まずセクション1が実行される。セクション1が実行されるとfork文によりスレッドが2本に分裂することから、次の実行候補としてセクション2とセクション4の2セクションが記憶される(状態1402)。ここで並行実行モードであればセクション2とセクション4が同時に選択され実行されるが、ここでは逐次実行モードで実行しているため、どちらか1個が選択されなければならない。そこでセクション2を選択し、実行したとする。セクション2が終了すると、次の候補としてのセクション3が記憶される(状態1403)。

【0348】次に、候補を見る限りではセクション3とセクション4のどちらを選択しても良さそうだが、先に示した実行順序のルール「セクション3はセクション4の後に実行される」により、セクション4しか選択することができないので、セクション4が実行されることになる。セクション4の実行が終了すると、それを実行したスレッドが終了するため候補記憶部1203ではセクション4に続くセクションが無くなっている(状態1404)。そして、セクション3を実行し全処理を終了したことになる(状態1405)。

【0349】本発明は、上記実施例に限定されない。

【0350】上記実施例は、それぞれ独立の実施例として記載したが、それぞれ適宜組み合わせることにより、より効果的な並行プログラムを作成することが可能になる。特に、実施例1～実施例8において作成された並行プログラムを実施例9の並行プログラム実行装置で実行させてその妥当性を検証することも可能であるし、実施例9において、ソースプログラムである並行プログラムを並行プログラム実行装置で実行させて、実施例1～実施例8に示すように、実行情報(テスト・デバッグ情報を含む)に基づいて、逐次プログラムを並行化する、或いは、ソースプログラムをバグのない並行プログラムに変換しても良い。

【0351】本実施例によれば、実行ログを意味を変えない範囲でユーザに理解し易い形に並べ替え、ユーザに提示することができる。並べ替えても意味が変わらないことが保証されているので、オリジナルの実行ログにバグがある場合は、並べ替え後の実行ログにおいてもバグが存在する。この性質により、テスト・デバッグの効率が向上するという効果が得られる。

【0352】(実施例10)本実施例では、図2の逐次化装置12の具体的な実現法について述べる。図117は逐次化装置12の実施例を示すブロック図である。この逐次化装置12は、テスト実行装置121、第1実行ログファイル記憶部122、解析装置123、先行関係

情報ファイル124、並べ替え装置125及び第2実行ログファイル記憶部126からなる。

【0353】図118は図117の逐次化装置12を用いた場合の並行プログラム作成方法の概略手順を示すフローチャートである。

【0354】(1) ステップA1:モデル化  
対象の並行システムに対し、並行性を用いた自然なモデル化を行う。また、各プロセス構造を決定する。更に、該プロセス内を並行プログラム等を用いたプログラミングにより、ソースプログラムCPを記述する。このソースプログラムには、バグが潜在的に存在する可能性がある。図119は、こうして記述されたソースプログラムCPの具体例であり、2つのプロセスP1とP2をプログラミングしている。

【0355】(2) ステップQ1:実行  
テスト実行装置121により、ソースプログラムCPを実行し、実行ログを第1実行ログファイル記憶部122に格納するとともに図2の出力装置5に表示する。この例の場合、図119のソースプログラムCPに対する可能な実行ログは数多く存在するが、ここでは、図120のような実行ログが生成されたとする。しかし、この実行ログはプロセスP1とプロセスP2が複雑に絡み合っており、実行過程が把握しにくいものとなっている。

【0356】(3) ステップQ2:バグの判定  
テストの結果、バグがなければ終了する。テストの結果バグがあれば、その実行ログ3を保存しステップQ3に進む。この例の場合、図120の実行ログが示す実行後のMの値は「M=-1」であり、バグであると判定したとする(期待した値は「M=0」)。

【0357】(4) ステップQ3:解析  
解析装置123により、第1CPファイル記憶部11からのソースプログラムCP及び第1実行ログファイル記憶部122に格納された実行ログからプロセス間の先行関係情報を抽出し、先行関係情報ファイル124に格納する。図121に、図119の実行ログにおける2つのプロセスP1とP2の各命令の先行関係を示す。

【0358】(5) ステップQ4:実行ログの並べ替え

第1実行ログファイル記憶部122と先行関係情報ファイル124に格納された実行ログ及び先行関係情報から、並べ替え装置125によりユーザが理解しやすい順序に並べ替えた実行ログを生成し、第2実行ログファイル記憶部126に格納する。この例の場合、図121の先行関係を満たす範囲で、ユーザが理解しやすい順序に並べ替えた実行ログを生成する。具体的な並べ替え規則の一例としては、(a)プロセスに優先順位を導入し、その優先順位を用いて実行ログを並べ替える、(b)待ち状態が解除されたプロセスを優先して実行ログを並べ替える

(c)ユーザが指定する、などがある。図122に、ブ

ロセスに優先順位を導入して並べ替えた後の実行ログを示す。

【0359】(6) ステップQ5：実行ログの表示  
第2実行ログファイル記憶部126に格納された並べ替え後の実行ログを逐次プログラムとして出力し、かつ出力装置5により表示してユーザに提示する。

【0360】この例では、図122に示す実行ログが出力装置5で図122に示すように表示される。この実行ログはプロセスP1とプロセスP2の命令が先行関係を保つ範囲でまとまっており、ユーザにとって実行過程が把握しやすいものとなっている。

【0361】(7) ステップQ6：並行プログラムのデバッグ／修正

ユーザは出力装置5で表示された実行ログを見て実行過程を把握又は実行を再現し、バグの原因を発見すればプログラムを修正する。プログラムを修正したらステップQ1に戻る。この例の場合、図122の実行ログの表示から、ユーザは、バグの原因は「P2：read(M、Y)：」と「P1：write(M、XX)：」の順序が間違っていることであると認識し、P2のsend命令の場所を修正することができる。修正されたプログラムは図123のようになる。この修正後のプログラムを実行すると、実行後のMの値は「M=0」となり、バグが削除できたことがわかる。

【0362】(実施例11) 本実施例では、テスト実行は部分超逐次プログラムに対して行うが、ソースコードの修正はオリジナルの並行プログラムに対して行う。また、良い非決定性に関する情報の導入は、逐次化条件に対して行い、良い非決定性の導入毎にオリジナルの並行プログラムの逐次化(部分逐次化)を行う。

【0363】図124は、本実施例に係る並行プログラム作成支援装置の概略構成を示す図である。同図において、第1CPファイル記憶部11に格納されたソースプログラムCPは、ユーザによる入力装置4からの指示により引き出され、逐次化装置12に入力される。逐次化装置12に入力されたソースプログラムCPは、逐次化ルール記憶部13に格納された逐次化ルールに従って超逐次プログラムHSP又は部分超逐次プログラムPHSPに変換され、HSP/PHSPファイル記憶部141に記録される。

【0364】このプログラムHSP/PHSPに対して、テスト実行装置15でテストが行われる。ここで、バグがあればデバッグ装置16を用いて、第1CPファイル記憶部11に格納されているソースプログラムCPを修正する。修正されたソースプログラムCPは再度逐次化され、超逐次プログラム(HSP)又は部分超逐次プログラム(PHSP)が生成されてHSP/PHSPファイル記憶部141に記録される。

【0365】この一連のサイクルを繰り返し、テスト・デバッグが完了したならば、非決定性導入装置17で、

逐次化ルール記憶部13に格納されている逐次化ルールの制約を緩めて、部分超逐次プログラムPHSPを生成する。そして、この部分超逐次プログラムPHSPに対して、同様のテスト・デバッグを繰り返しながら、非決定性導入装置17で、徐々に逐次化ルールの制約を弱めていく。このようにして良い非決定性導入が完了したならば、最終的に並行化装置18で並行プログラムが生成され、第2CPファイル記憶部19に格納される。

【0366】(実施例12) 本実施例では、テスト超逐次プログラムに対して行うが、バグのあった場合はソースコードたる並行プログラムを修正せずに、逐次化ルールを修正することでテスト・デバッグを行う。その他の部分に関しては、基本的にこれまでの実施例と同じである。

【0367】本実施例における並行プログラムの作成は、以下のような手順によって実現される。図125は、本実施例に係る並行プログラム作成方法の概略手順を示すフローチャートである。

【0368】(1) ステップA1：モデル化  
対象の並行システムに対し、並行性を用いた自然なモデル化を行う。また、各プロセス構造を決定する。更に、該プロセス内を並行プログラム等を用いたプログラミングにより、並行構造を有する並行プログラムをソースプログラムとして記述する。このソースプログラムには、バグが潜在的に存在する可能性がある。

(2) ステップA2：逐次化  
ステップA1で得られた並行プログラムを所定の逐次化ルールに基づいて超逐次プログラムに変換する。

【0369】(3) ステップA3：テスト・デバッグ  
ステップA2で変換された超逐次プログラムを実行してテストを行い、機能面のバグがある場合は、超逐次プログラムを修正してテスト・デバッグを行う。

【0370】(4) ステップC2：バグ判定  
ステップA3のテストの結果、タイミング面のバグがある場合は、ステップJ1に移行する。いずれのバグもない場合は、その超逐次プログラムを保存しステップA4に移行する。

【0371】(5) ステップJ1：逐次化ルールの修正  
ステップC2でタイミング面のバグがあると判定された場合は、逐次化ルールを修正し、ステップA2に戻る。

【0372】(6) ステップA4：良い非決定性の導入

良い非決定性の追加が必要な場合は、ステップJ1により逐次化ルールを修正して新たな逐次化ルールを設定し、ステップA2に戻る。良い非決定性の追加が必要でない場合は、ステップA7に移行する。

【0373】(7) ステップA7：並行化コンパイル  
保存された超逐次プログラムの集合から、無害な非決定性部分を抽出し、それを並行プログラムとして生成す

る。

【0374】図126は、本実施例に係る並行プログラム作成支援装置の概略構成を示す図である。同図において、第1CPファイル記憶部11に格納されているソースプログラムCPは、逐次化装置12において逐次化ルール記憶部13に格納された逐次化ルールに従って、メタレベルにおいてデフォルト逐次性が導入されることにより、超逐次プログラム(HSP)に変換され、HSPファイル記憶部14に記録される。ユーザは、この超逐次プログラムHSPを出力装置5によって見ることができる。この超逐次プログラムHSPは、ユーザによる入力装置4からの指示によってテスト実行装置15に入力され、テスト実行が行われる。

【0375】テスト実行装置15は、テスト実行の結果(実行ログ)を出力装置5に提示する。ユーザは、このテスト実行の結果に基づいて、逐次化ルール修正装置26を用いて逐次化ルール記憶部13に格納されている逐次化ルールを修正し、超逐次プログラムHSPに対し所定のテスト・デバッグを行うことができる。

【0376】ユーザは、このように逐次化ルールの修正により超逐次プログラムHSPに対して所定のテスト・デバッグを行った後、入力装置4から再度テスト実行の指示を与える。これにより、テスト・デバッグの行われた超逐次プログラムHSPは、テスト実行装置15に入力され、再度テスト実行が行われる。このテスト・デバッグは、超逐次プログラムHSPが正常に動作することを確認するまで繰り返し行われる。

【0377】そして、本実施例ではテスト・デバッグにより正常に動作することを確認した時点で、逐次化ルール修正装置26により逐次化ルール記憶部13に格納されている逐次化ルールが修正され、この修正の後の逐次化ルールが逐次化ルール記憶部13に新たに記録される。この逐次化ルールの修正によって、超逐次プログラムに対して良い非決定性に関する情報が部分的に導入される。

【0378】その他本発明の要旨を変更しない範囲で種々変形して実施できるのは勿論である。

【0379】

【発明の効果】本発明によれば次のような効果が得られる。

【0380】本発明では並行プログラムを一旦逐次化し、逐次化したプログラムに対してテスト・デバッグを行うことにより、従来の並行プログラムのプログラミングより遥かに容易な逐次プログラミングと同じレベルの困難さで並行プログラムのテスト・デバッグが可能となる。

【0381】上記のグラフ情報によって並行化及び逐次化の情報をユーザに対して同時に提示することにより、ユーザは第1並行プログラムの並行構造を考慮しつつ、良い非決定性部分の指定をすることができるようにな

る。また、並行プログラム記述レベルにおける良い非決定性部分の指定・解除ではなく、グラフ情報に対して良い非決定性部分を指定・解除を行うことができるため、高度な並行プログラミング技術を必要とすることなく、容易に並行プログラムの開発をすることができるようになる。

【0382】第1並行プログラムを逐次プログラムに変換する際、第1並行プログラム及びその実行ログを解析し、この解析結果に基づいて実行ログを並べ替えるようにすれば、並べ替え後の実行ログを表示してユーザに提示することによって、並行プログラムの実行過程の理解が容易となり、テスト・デバッグの効率が向上する。逐次プログラムに並行性に関する情報を導入する際、逐次プログラムのプロセスの流れを制約と遷移条件からなるフィールドに変換し、更にそのフィールドデータを表示することによって、フィールドを対話的・視覚的に編集することで並行性に関する情報を効果的に導入し、バグのない並行プログラムが効率的に作成される。

【0383】第1並行プログラムから逐次化された逐次プログラムに対して並行化の候補となるプロセス群を指定し、このプロセス群の実行順序を入れ換えて逐次プログラムを複数の並行模擬プログラムに変換した後、これら複数の並行模擬プログラムを部分的に逐次構造を有する1つの部分逐次プログラムに変換し、この部分逐次プログラムを並行化して第2並行プログラムに変換することにより、部分逐次プログラム上で並行プログラムの動作を十分に確認することができる。また、部分逐次プログラムに対して並行化の候補となるプロセス群を指定することで、逐次構造プログラムを段階的に並行プログラムへ変換することができる。更に、並行模擬動作系列に基づく逐次実行で正しく動作することが確認された非決定性のみを許容する並行性に関する情報を導入することで、正しく動作する並行プログラムを得ることができる。これらによって、並行プログラムのテスト・デバッグが容易となる。

【0384】プロセス群がメッセージ情報を交換しながら並行して動作する実行環境に用いられる並行プログラムの作成を支援する並行プログラム作成支援装置において、第1並行プログラムのプロセス群の実行履歴であるログ情報を逐次化ルールとして取得して記憶し、このログ情報を修正可能とするとともに、記憶されているログ情報に基づいてこれに基づいてプロセス群を逐次的に起動制御し、記憶されたログ情報を並行化して第2並行プログラムに変換することにより、ソースプログラムとしての並行プログラムを修正することなく、ログ情報の修正で処理タイミングの非決定性に起因する不具合を解決できる。これにより、処理タイミングの非決定性が内在する並行／並列／分散プログラムの開発が容易となる。また、ユーザの意図する良い非決定性のみを容易に導入することができるため、並行プログラムとしての柔軟

性、再利用性及び拡張性を維持することもできる。

【0385】プロセス群がメッセージ情報を交換しながら並行して動作できる実行環境でプロセス群が実行順序規定情報に従って動作するシステムにおいて、実行順序規定情報を分割し、つまり並行プログラムを逐次化して得られた全プロセスの集中ログ情報を各プロセス毎に分割し、この分割された実行順序情報に基づいてプロセス群を起動制御することにより、無害の非決定性を自然に導入し、集中ログ情報に基づく逐次プログラムの実行時と同一結果を高い処理効率で得ることが可能となる。

【0386】第1並行プログラムをテスト実行してそのテスト実行の結果、その1つであるバグのない実行ログを蓄積し、この蓄積されたバグのない実行ログのみを並行化して第2並行プログラムに変換することにより、テストで通過したタイミングだけを許容するようにプログラムが動くようになるため、テストしなかったことで残存したバグに陥ることを回避でき、信頼性が向上する。

【0387】本発明の並行プログラム実行装置によれば並行プログラムの逐次又は部分並行実行を行なうことが可能となるので、従来の逐次プログラムと同様に再現性を保証することができ、効果的かつ安定したテスト・デバッグを行なうことができる。また、逐次実行から部分並行実行に容易に切替えることができるため、逐次化してテスト・デバッグをおこなったプログラムを非決定性に影響されない安全な並行実行を可能とする。他に、セクションの実行順序に関するルールを編集して実行時のセクション間の同期を制御することにより、効果的なテスト・デバッグを行なえたとともにプログラムの効率化の指針とすることもできる。

#### 【図面の簡単な説明】

【図1】 本発明に係る並行プログラム作成支援装置を実現するためのコンピュータシステムの構成図。

【図2】 実施例1に係る並行プログラム作成支援装置の概略構成を示すブロック図。

【図3】 実施例1に係る並行プログラム作成方法の概略手順を示すフローチャート。

【図4】 実施例1の動作説明のための並行プログラムの一例と逐次化ルールを示す図。

【図5】 実施例1の動作説明のためのメタレベルのデフォルト逐次性が導入された超逐次プログラムの概念図。

【図6】 実施例1の動作説明のためのデバッグ画面の一例を示す図。

【図7】 実施例1の動作説明のための非決定性に関する情報の導入例を示す図。

【図8】 実施例2に係る並行プログラム作成支援装置の概略構成を示すブロック図。

【図9】 実施例2に係る並行プログラム作成方法の概略手順を示すフローチャート。

【図10】 セクションの設定方法の一例を示す図。

【図11】 逐次化方式の一例を示す図。

【図12】 逐次化情報の書換ルール例を示す図。

【図13】 並行プログラムの例を示す図。

【図14】 並行プログラムの記載例を示す図。

【図15】 それぞれ、超逐次プログラムのセクション情報、プログラム構造情報、超逐次化情報を示す図。

【図16】 良い並行性の導入を説明するための図。

【図17】 自動並行化を行った逐次プログラミングから生成された並行プログラムのソースコードを示す図。

【図18】 超逐次プログラムのプログラム構造情報を示す図。

【図19】 超逐次プログラムの逐次化情報を示す図。

【図20】 自動並行化された超逐次プログラムの処理の流れを示す図。

【図21】 実施例3に係る並行プログラム作成支援装置の概略構成を示すブロック図。

【図22】 実施例3に係る並行プログラム作成方法の概略手順を示すフローチャート。

【図23】 実施例3の動作説明のためのソースプログラムである並行プログラムの一例を示す図。

【図24】 実施例3における並行プログラム合成例を示す図。

【図25】 実施例4に係る並行プログラム作成法の概略手順を示すフローチャート。

【図26】 実施例4で用いるプロセステーブルを示す図。

【図27】 実施例4の動作説明のための並行プログラムの一例を示す図。

【図28】 図27における並行プログラムの並行構造を示す概念図。

【図29】 実施例4で用いる並行プログラムの構造解析の結果作成されるプロセステーブルを示す図。

【図30】 実施例4で用いる超逐次グラフの一例を示す図。

【図31】 実施例4における超逐次グラフ生成装置の動作処理を説明するための図。

【図32】 実施例4における並行化部分の指定方法を説明するための図。

【図33】 実施例4における並行化部分指定後の超逐次グラフの一例を示す図。

【図34】 グループ化されたノードを持つ超逐次グラフの一例を示す図。

【図35】 優先順位変更後の超逐次グラフの一例を示す図。

【図36】 3プロセス間の良い非決定性部分の導入を説明するための図。

【図37】 良い非決定性部分の導入後の超逐次グラフを示す図。

【図38】 実施例5に係る並行プログラム作成支援装置の概略構成を示すブロック図。

【図 39】 実施例 5 の動作説明のためのソースプログラムである並行プログラムを示す図。

【図 40】 図 39 の並行プログラムで実行される処理の流れを示す図。

【図 41】 実施例 5 における逐次化プロセスファイルに格納されている逐次化プロセスの一例を示す図。

【図 42】 図 41 の逐次化プロセスで実行される処理の流れを示す図。

【図 43】 実施例 5 におけるフィールドデータ生成部で実行される処理の流れを示すフローチャート。

【図 44】 図 43 の処理により生成されるエリアのデータ構造を示す図。

【図 45】 実施例 5 におけるフィールドデータ生成部で生成されるフィールドデータの一例を示す図。

【図 46】 実施例 5 におけるフィールドチューニング部で実行される処理の流れを示すフローチャート。

【図 47】 図 46 の一部（ステップ E9）を詳細に示す図。

【図 48】 実施例 5 における制約変更操作によって起こるフィールドの変化を示す図。

【図 49】 図 46 の他の一部（ステップ E11）を詳細に示す図。

【図 50】 実施例 5 における制約変更操作によって起こるフィールドの変化を示す図。

【図 51】 図 46 の更に他の一部（ステップ E13）を詳細に示す図。

【図 52】 実施例 5 における制約変更操作によって起こるフィールドの変化を示す図。

【図 53】 フィールドに矛盾を発生させる自明な制約を検出する処理を示す図。

【図 54】 自明制約を含むフィールド例を示す図。

【図 55】 自明制約の検出例を示す図。

【図 56】 フィールドエディタの表示画面の例を示す図。

【図 57】 フィールドエディタの表示画面の他の例を示す図。

【図 58】 フィールドエディタの表示画面の更に他の例を示す図。

【図 59】 実施例 5 におけるフィールドデータのチューニング過程の様子を示す図。

【図 60】 実施例 5 におけるフィールドデータのチューニング過程の様子を示す図。

【図 61】 実施例 5 におけるフィールドデータのチューニング過程の様子を示す図。

【図 62】 実施例 5 におけるフィールドデータのチューニング過程の様子を示す図。

【図 63】 実施例 5 におけるフィールドデータのチューニング過程の様子を示す図。

【図 64】 実施例 5 におけるフィールドデータのチューニング過程の様子を示す図。

【図 65】 実施例 5 におけるフィールドデータのチューニング過程の様子を示す図。

【図 66】 実施例 5 におけるフィールドデータのチューニング過程の様子を示す図。

【図 67】 実施例 5 におけるフィールド変換部で実行される処理を示すフローチャート。

【図 68】 実施例 5 における修正後の並行プログラムの一例を示す図。

【図 69】 図 68 の並行プログラムで実行されるプロセスの流れをイメージした図。

【図 70】 実施例 6 に係る並行プログラム作成支援装置の概略構成を示す図。

【図 71】 実施例 7 に係る並行プログラム作成方法の概略手順を示すフローチャート。

【図 72】 実施例 7 の動作説明のためのユーザが想定した並行プログラムモデルの一例を示す図。

【図 73】 図 72 の並行プログラムから得られる超逐次プログラムの実行系列をモデル化した例を示す図。

【図 74】 実施例 7 における超逐次プログラムから部分並行プログラムへの変換手順を示すフローチャート。

【図 75】 実施例 7 における超逐次プログラムから部分並行プログラムへの変換例を示す図。

【図 76】 実施例 7 における超逐次プログラムから部分並行プログラムへの変換例を示す図。

【図 77】 実施例 7 における超逐次プログラムから部分並行プログラムへの変換例を示す図。

【図 78】 実施例 7 における超逐次プログラムから部分並行プログラムへの変換例を示す図。

【図 79】 実施例 8 に係るプロセス間の情報交換の一例を示す図。

【図 80】 実施例 8 に係る並行プログラム作成支援装置の概略構成を示すブロック図。

【図 81】 実施例 8 に係る並行プログラム作成方法の概略手順を示すフローチャート。

【図 82】 実施例 8 におけるメッセージの実行履歴であるログ情報の一例を示す図。

【図 83】 実施例 8 に係るログ情報に基づく並行プログラムデバッグ装置の概略構成を示すブロック図。

【図 84】 実施例 8 におけるメッセージフォーマットの一例を示す図。

【図 85】 実施例 8 におけるプロセス間情報交換を示す図。

【図 86】 実施例 8 におけるログ情報の一例を示す図。

【図 87】 実施例 8 におけるプロセス群制御部の処理手順を示すフローチャート。

【図 88】 実施例 8 におけるログ情報を修正するために入れ替え対象を指示している一例を示す図。

【図 89】 実施例 8 における修正後のログ情報の一例を示す図。



【図90】 実施例8におけるログ情報に良い非決定性を導入するためにその対象箇所を指定している一例を示す図。

【図91】 実施例8における良い非決定性導入後のログ情報の一例を示す図。

【図92】 実施例8におけるログ情報に基づく並行プログラム実行システムの構成を示す図。

【図93】 実施例8におけるタスク間情報交換を示す図。

【図94】 実施例8における良い非決定性を導入した場合のログ情報の一例を示す図。

【図95】 実施例8におけるタスク制御部の処理手順を示すフローチャート。

【図96】 実施例8におけるタスク制御部を起動するための一手段であるプログラム例を示す図。

【図97】 実施例8におけるメッセージフォーマットを示す図。

【図98】 実施例8におけるタスク制御部の処理手順を示すフローチャート。

【図99】 実施例8における各タスクの構成例を示す図。

【図100】 セクションの概念を示す図。

【図101】 セクションの融合を示す概念図。

【図102】 セクションの分割を示す概念図。

【図103】 本発明の実施例9の概略構成を示す図。

【図104】 セクションを記憶するためのデータ構造を示す図。

【図105】 セクション管理テーブルの構造を示す図。

【図106】 部品テーブルの構造を示す図。

【図107】 セクションの実行順序のルールを説明するための図。

【図108】 プログラム解析部の詳細構成を示す図。

【図109】 プログラム解析の流れを示すフローチャート。

【図110】 セクション情報編集部の詳細構成を示す図。

【図111】 コンパイル部の詳細構成を示す図。

【図112】 実行部の詳細構成を示す図。

【図113】 実行部の処理の流れを示すフローチャート。

【図114】 実行セクション候補記憶部の変化を説明するための図。

【図115】 実行セクション候補記憶部の変化を示す図。

【図116】 従来技術の問題点を説明するための図。

【図117】 実施例10に係る逐次化装置の構成を示すブロック図。

【図118】 実施例10に係る並行プログラム作成方法の概略手順を示すフローチャート。

【図119】 実施例10の動作説明のためのソースプログラムである並行プログラムの一例を示す図。

【図120】 実施例10の動作説明のための図119の並行プログラムの実行ログを示す図。

【図121】 実施例10の動作説明のための並行プログラムの先行関係を示す図。

【図122】 実施例10の動作説明のための並行プログラムの実行ログの並べ替えた後の状態を示す図。

【図123】 実施例10の動作説明のための修正後の並行プログラムの一例を示す図。

【図124】 実施例11に係る並行プログラム作成支援装置の概略構成を示す図。

【図125】 実施例12に係る並行プログラム作成方法の概略手順を示すフローチャート。

【図126】 実施例12に係る並行プログラム作成支援装置の概略構成を示すブロック図。

【符号の説明】

1～1～1-N…プロセッサ	2…I/Oインタフェース
3…共有メモリ	4…入力装置
5…出力装置	6…記憶装置
11…第1CPファイル記憶部	12…逐次化装置
13…逐次化ルール記憶部	14…HSPファイル記憶部
15…テスト実行装置	16…デバッグ装置
17…非決定性導入装置	18…並行化装置
19…第2CPファイル記憶部	
20…解析情報記憶部	21…並行化ルール記憶部
22…実行装置	23…分割基準指定装置
25…変更装置	26…逐次化ルール修正装置
31…逐次化情報記憶部	32…逐次化情報読込部
33…並行構造解析部	34…逐次構造解析部
35…画像データ生成部	
51…逐次化プロセスリスト記憶部	
52…フィールドデータ記憶部	53…フィールドデータ生成部
54…フィールドチューニング部	
55…フィールドエディタ	
201…ログ情報記憶部	202…ログ情報取得部
203…プロセス群制御部	204…メッセージ保存部
205…ログ情報修正部	
301…集中ログ情報記憶部	302…ログ情報分割部
303…分割基準指定部	304…分割ログ情報

## 記憶部

305...プロセス制御部

セージ保存部

401...テスト実行装置

記憶部

403...実行ログ記憶部

306...処理待ちメッ

402...テストケース

404...実行ログデー

## データベース

411...プログラム解析部

報記憶部

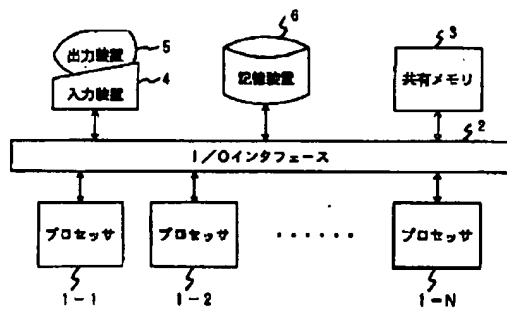
413...セクション情報編集部

415...実行部

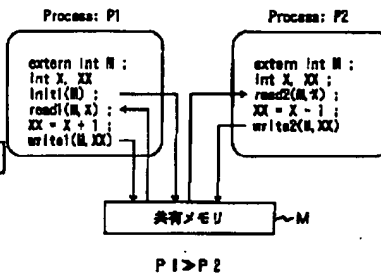
412...セクション情

414...コンパイル部

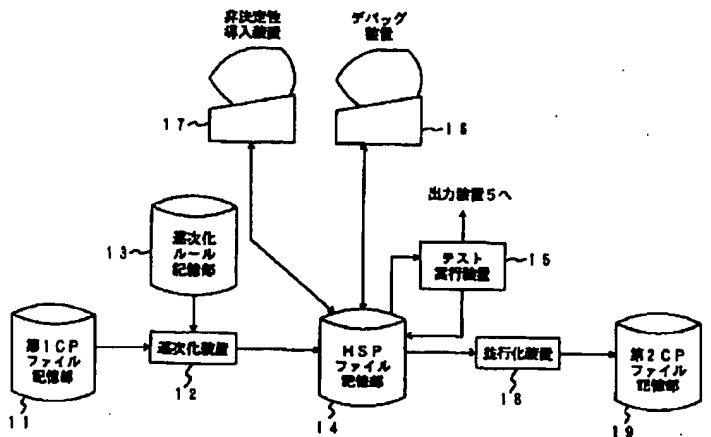
【図1】



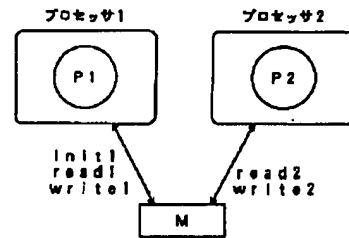
【図4】



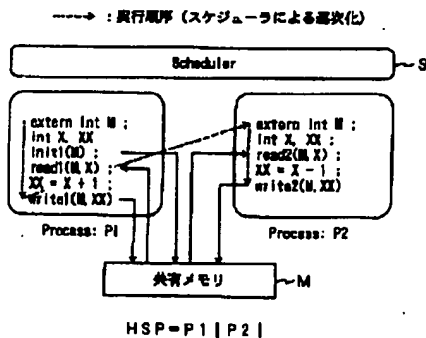
【図2】



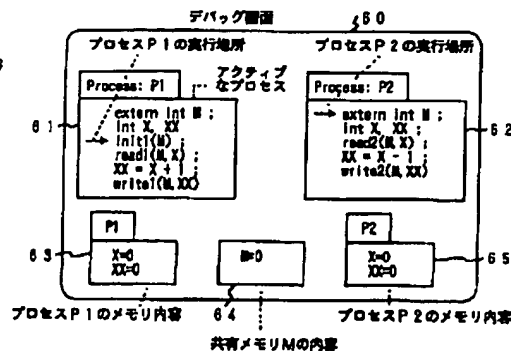
【図13】



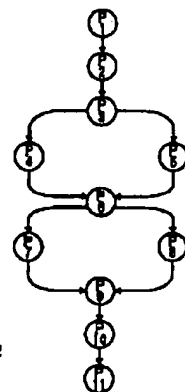
【図5】



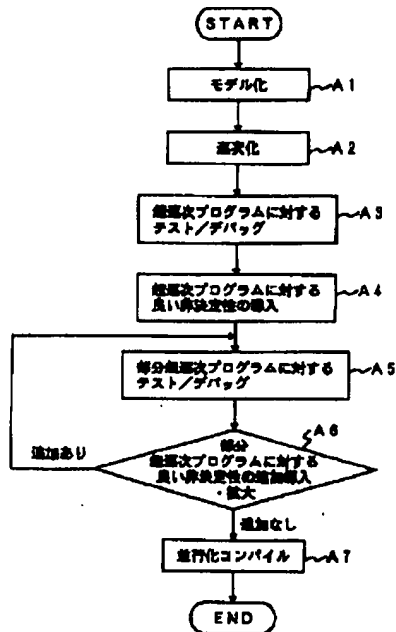
【図6】



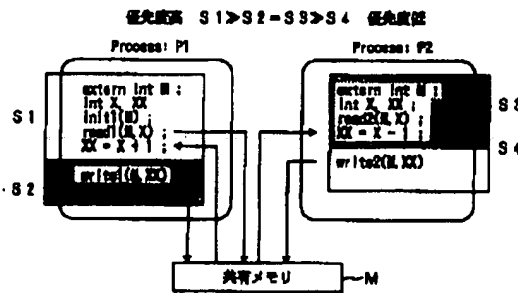
【図40】



【図3】



【図7】



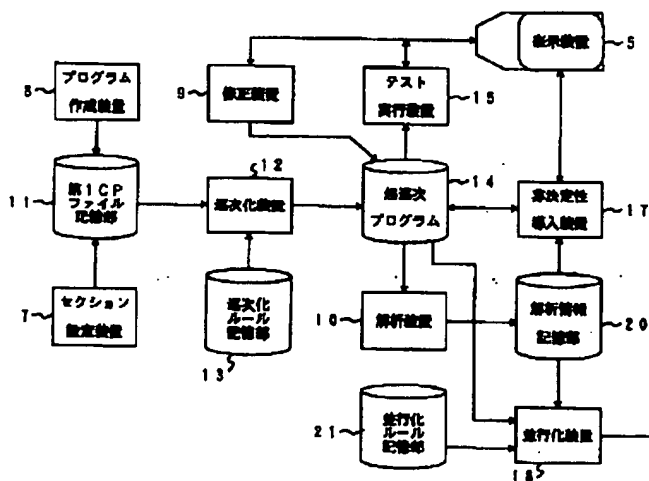
【図14】

```

P1:
begin
  Init1;      /*メモリを初期化する*/
  read1;     /*メモリから読み込む*/
  write1;    /*メモリに書き込む*/
end

P2:
begin
  read2;     /*メモリから読み込む*/
  write2;    /*メモリに書き込む*/
end
  
```

【図8】



【図17】

```

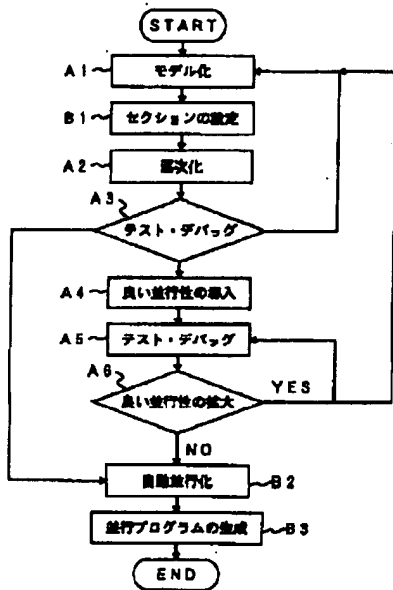
P1:
begin
  Init1;      /*メモリを初期化する*/
  send(1);   /*セマフォ*/
  read1;     /*メモリから読み込む*/
  write1;    /*メモリに書き込む*/
  send(2);  /*セマフォ*/
end

P2:
begin
  wait(1);   /*セマフォ*/
  read2;     /*メモリから読み込む*/
  wait(2);   /*セマフォ*/
  write2;    /*メモリに書き込む*/
end
  
```

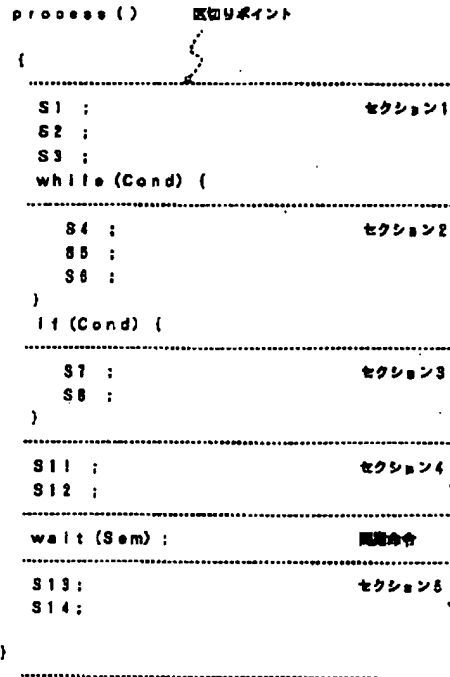
【図26】

§F1	§F2	§F3	§F4	§F5
電計 フィールド	ポインタ フィールド	優先順位 フィールド	優先順位用 バッファ フィールド	グループ化 情報 フィールド

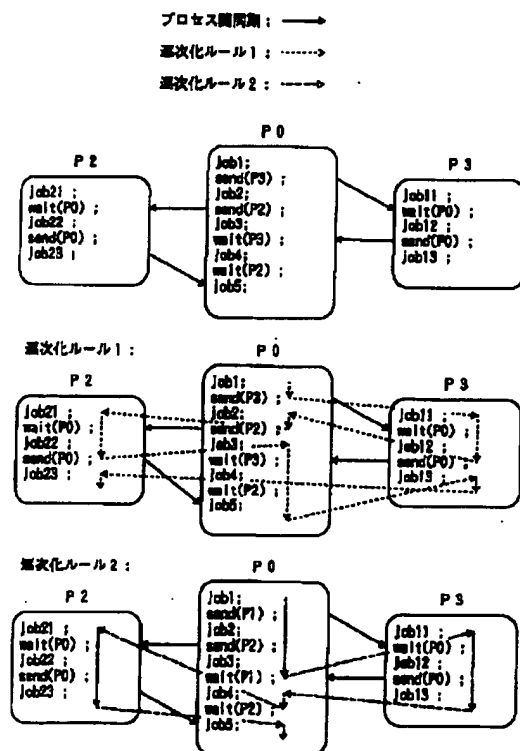
【図9】



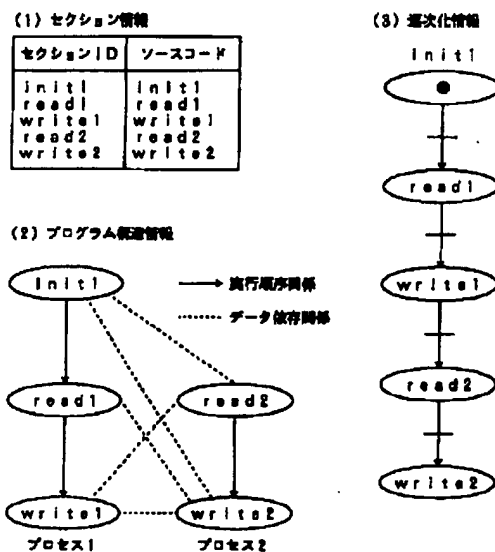
【図10】



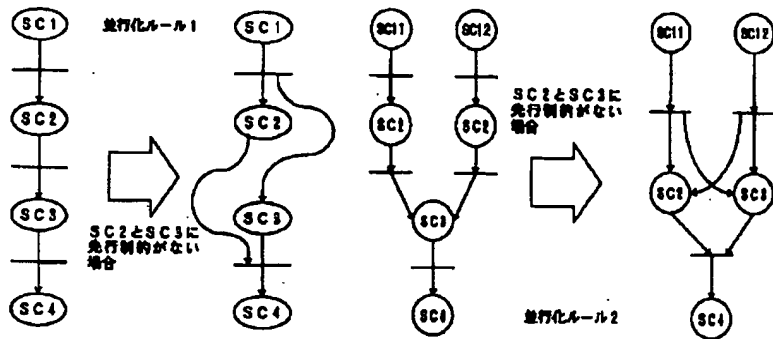
【図11】



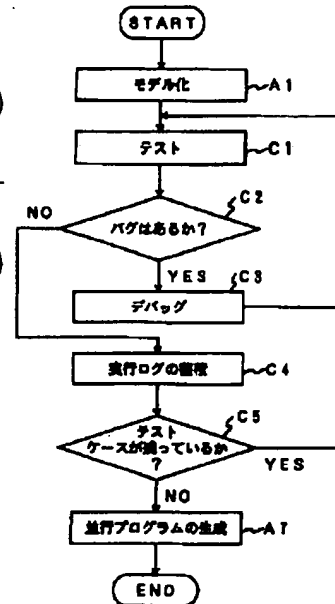
【図15】



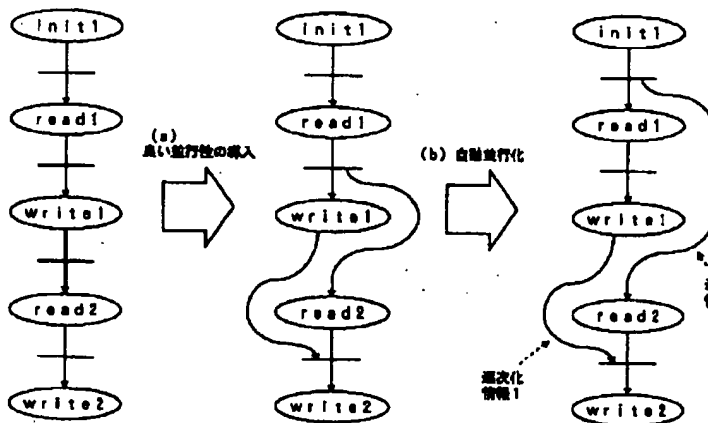
【図12】



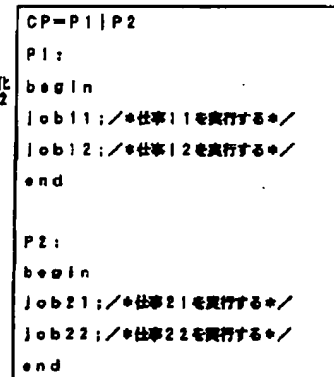
【図22】



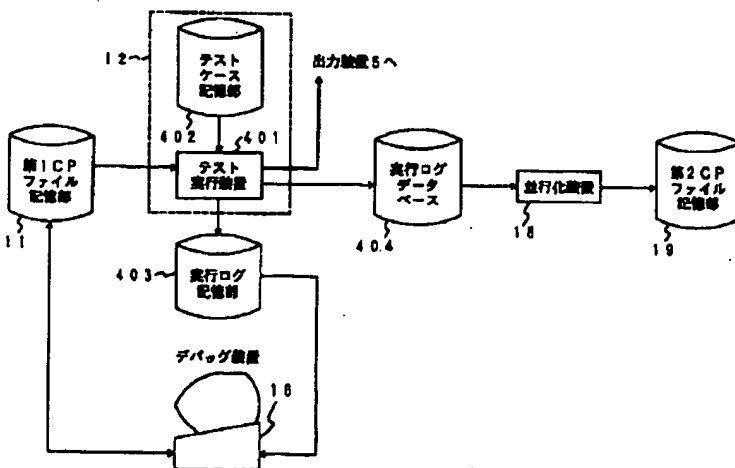
【図16】



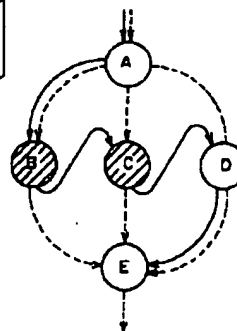
【図23】



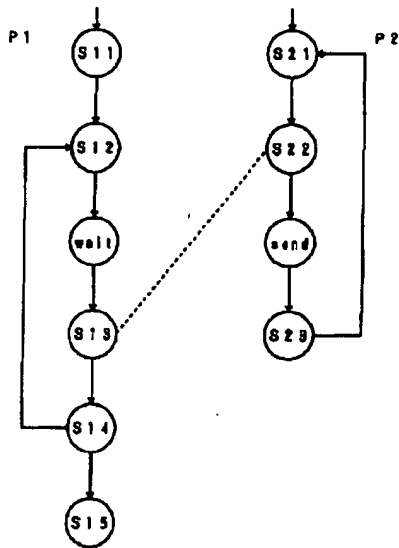
【図21】



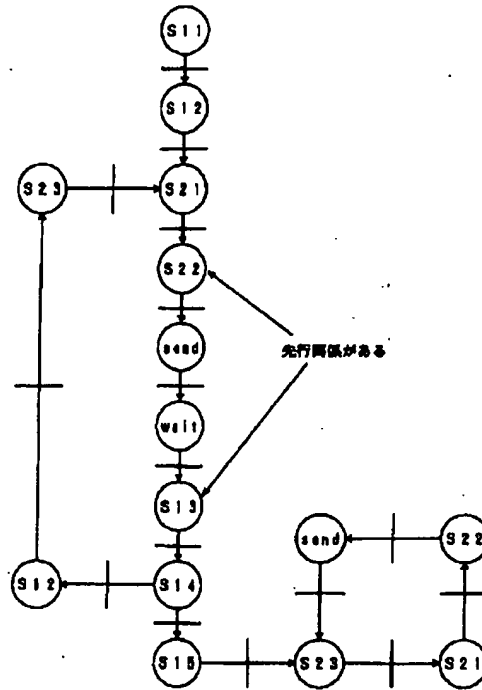
【図36】



【図 18】



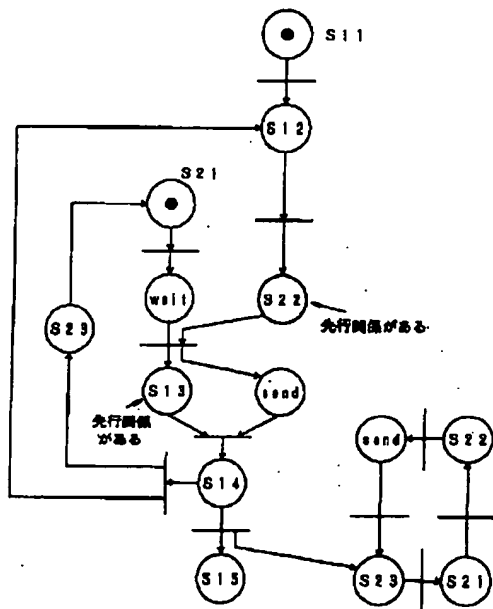
【図 19】



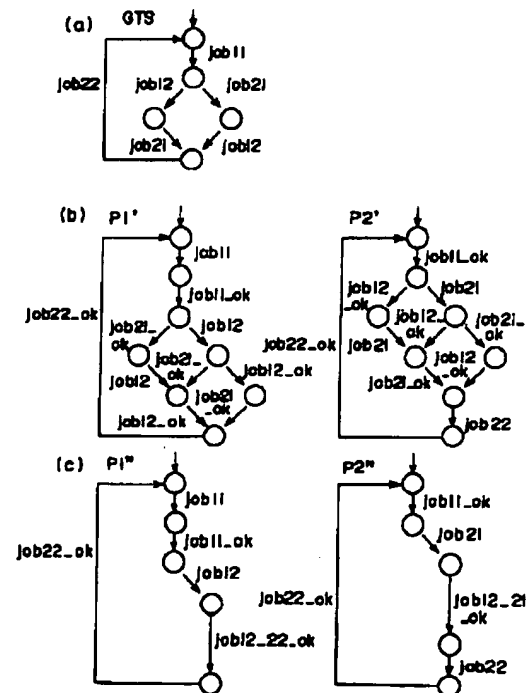
【図 22】



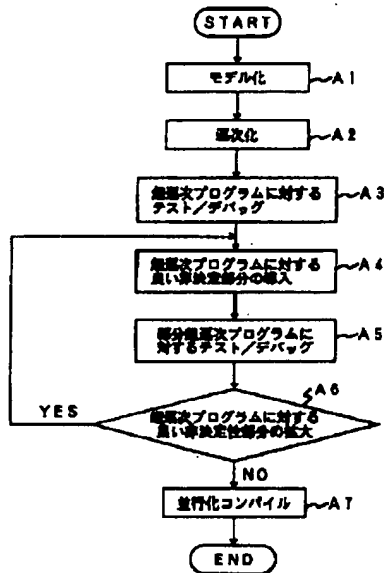
【図 20】



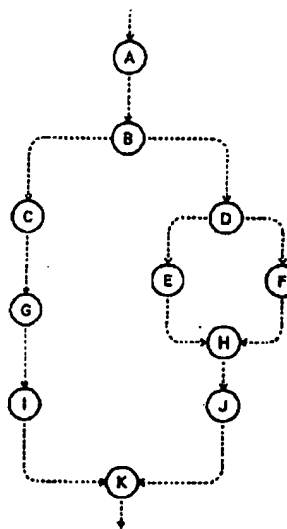
【図 24】



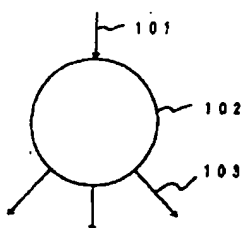
【図25】



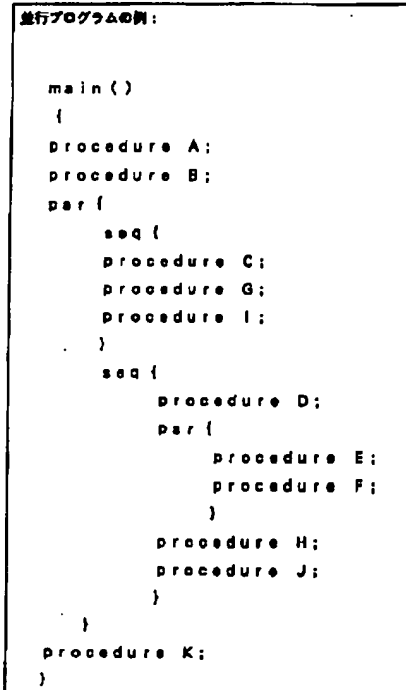
【図28】



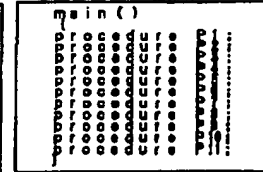
【図100】



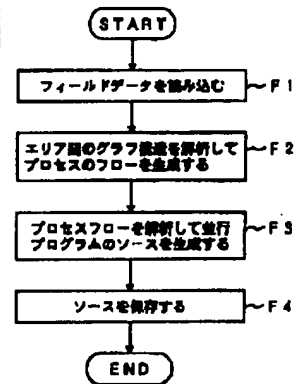
【図27】



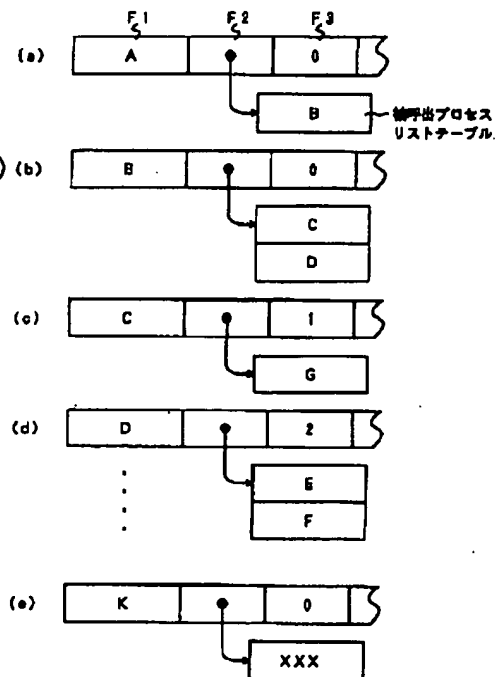
【図41】



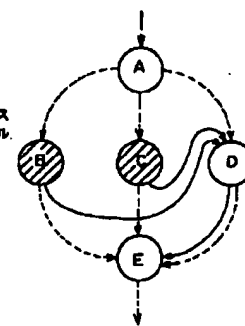
【図67】



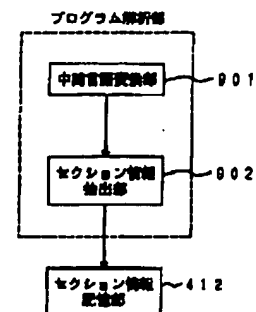
【図29】



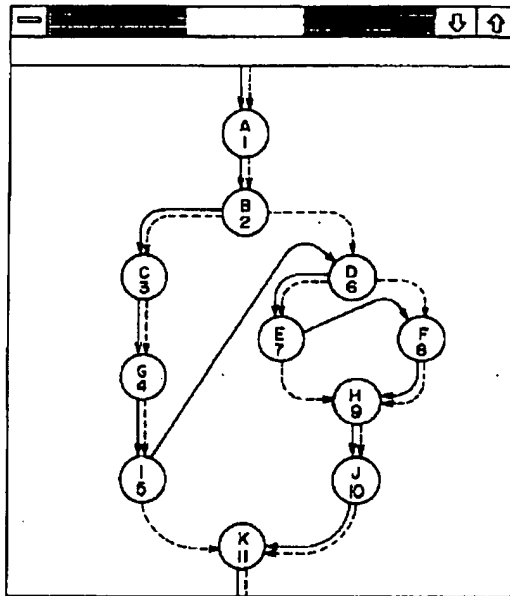
【図37】



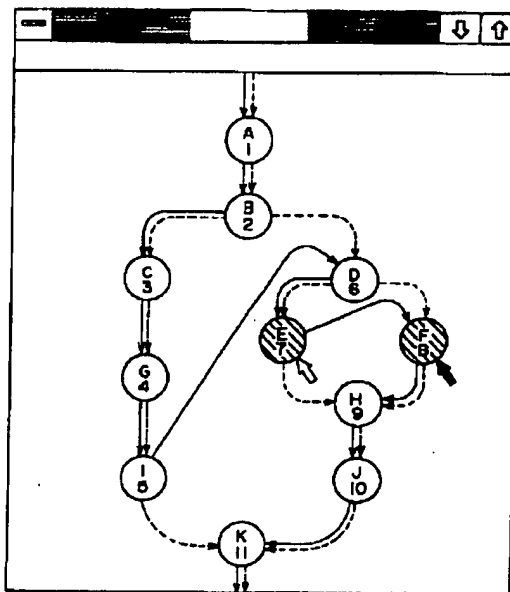
【図108】



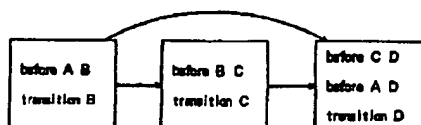
【 30 】



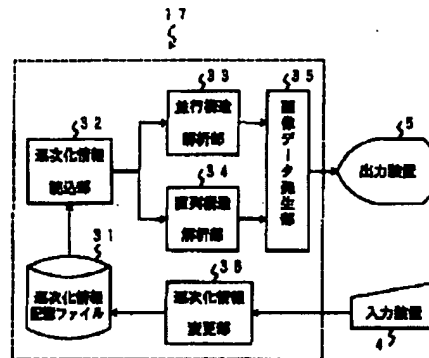
【圖 3 2】



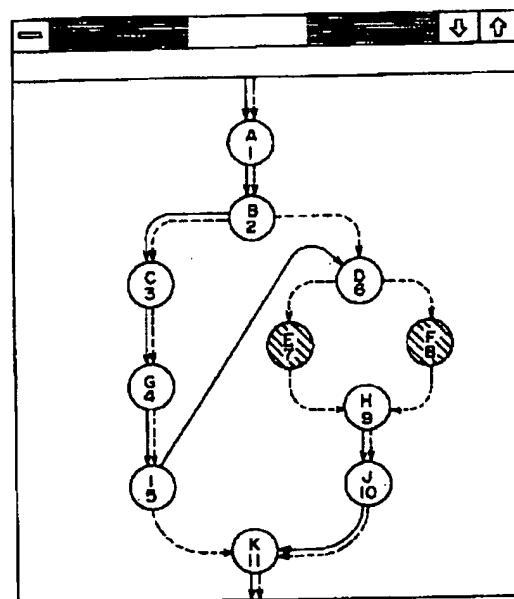
【圖 5 4】



【圖 3 1】



【圖 33】

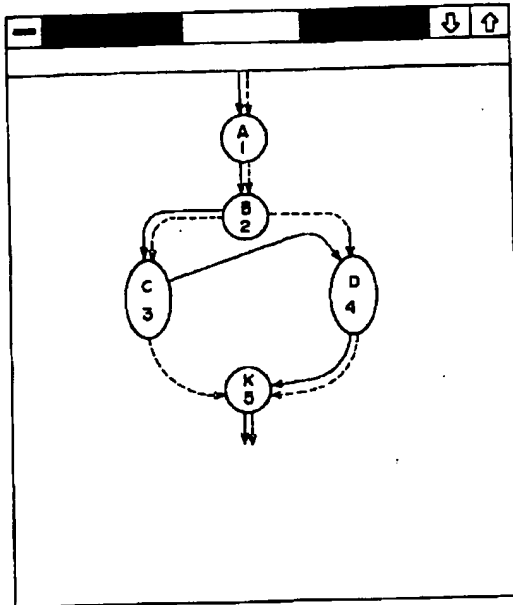


【圖 39】

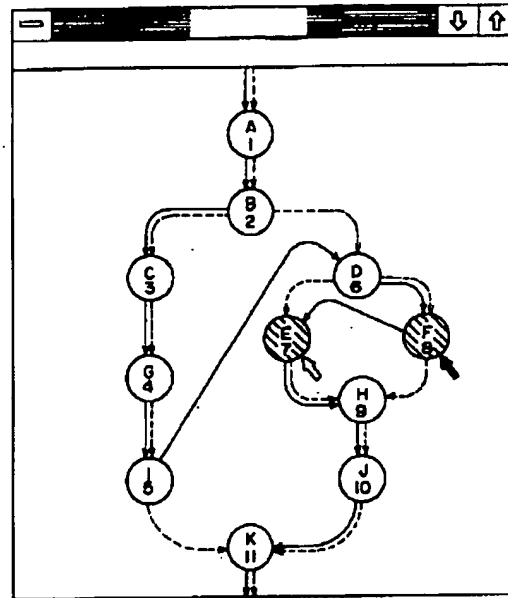
```
main()
  procedure P1;
  begin
    procedure P2;
    procedure P3;
    procedure P4;
  end;
  procedure P5;
  procedure P6;
  procedure P7;
  procedure P8;
  procedure P9;
end;
```



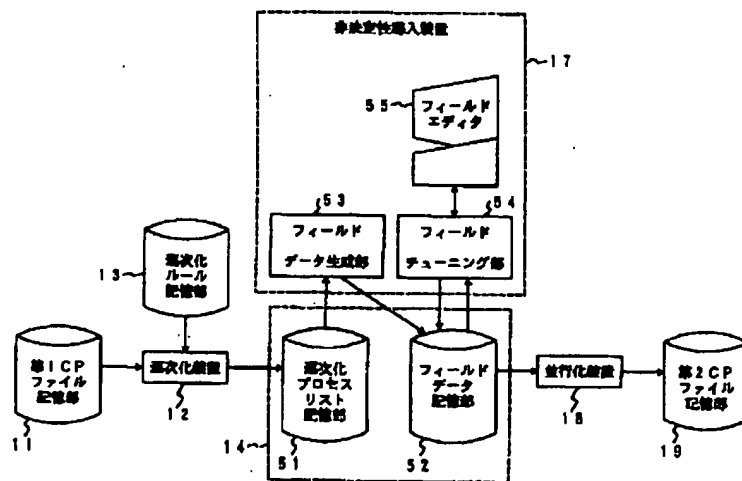
【図34】



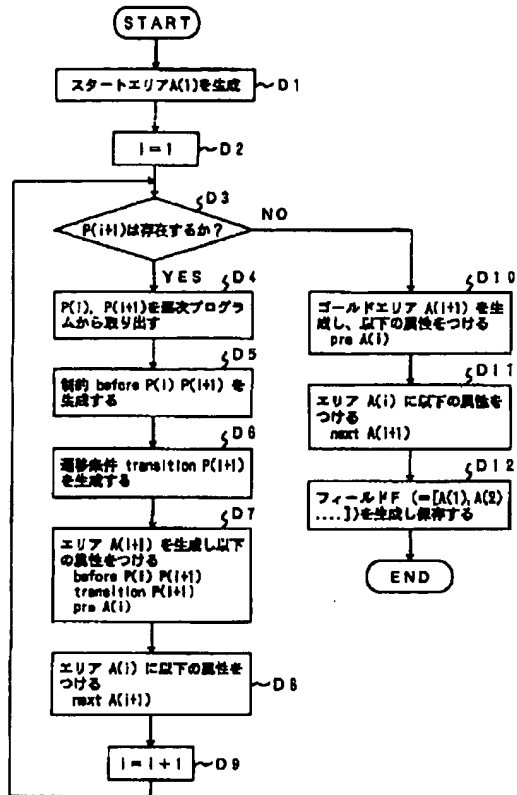
【図35】



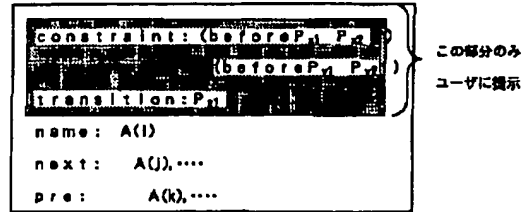
【図38】



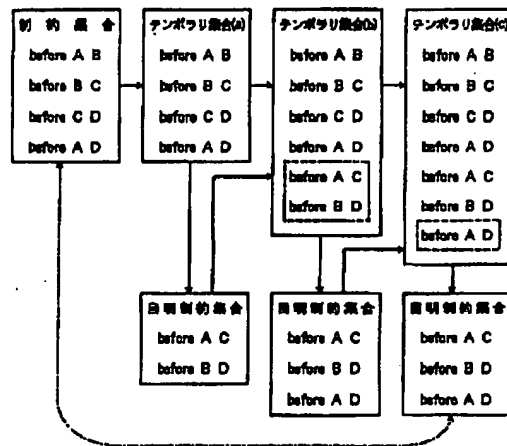
【図43】



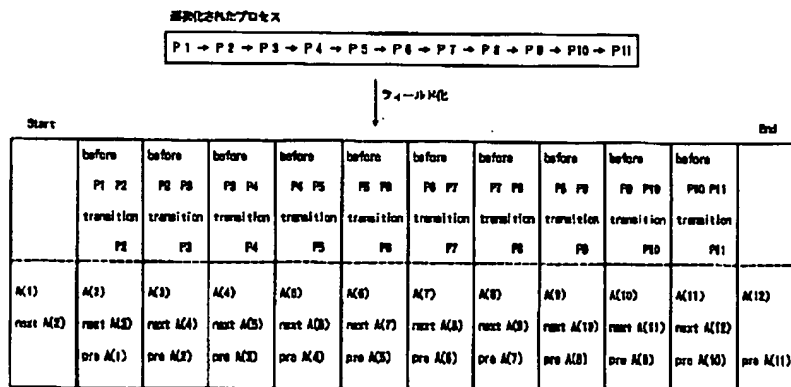
【図44】



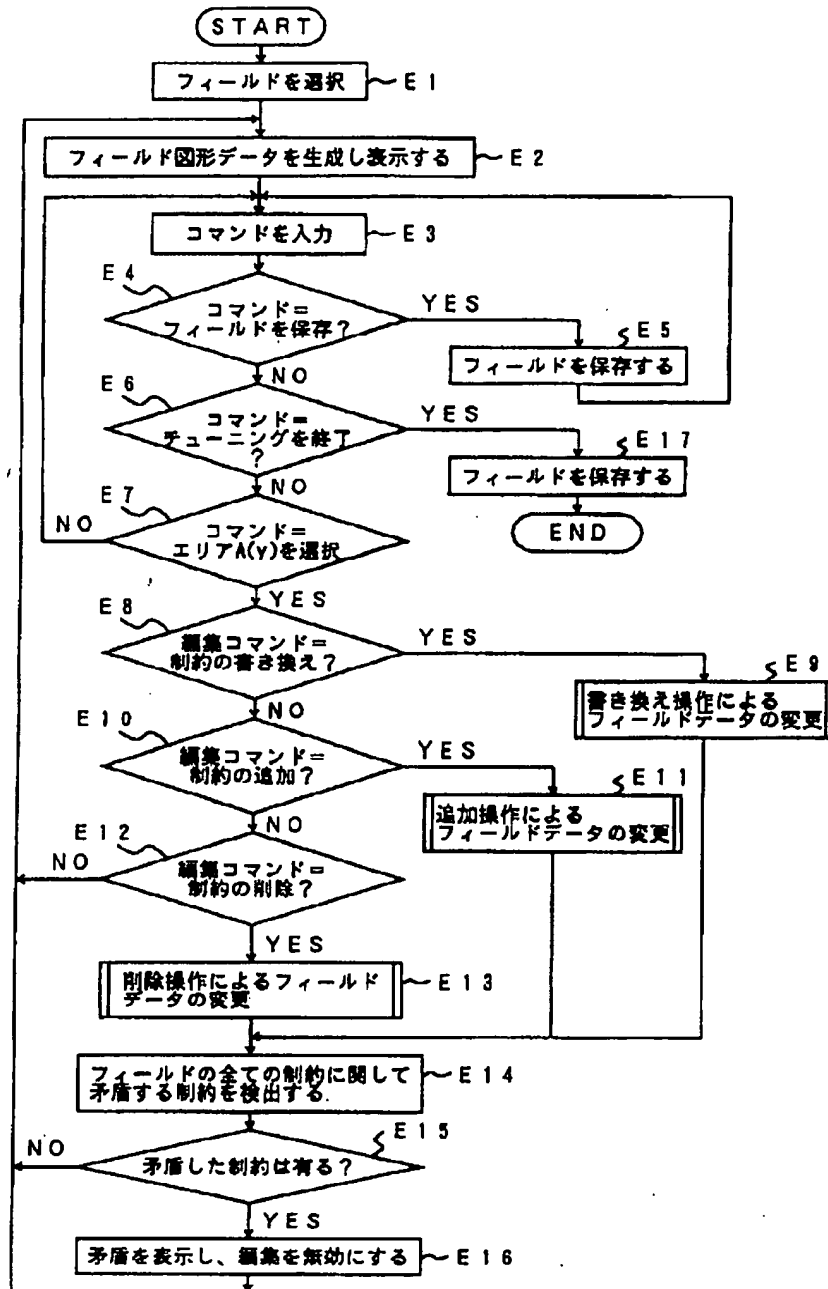
【図55】



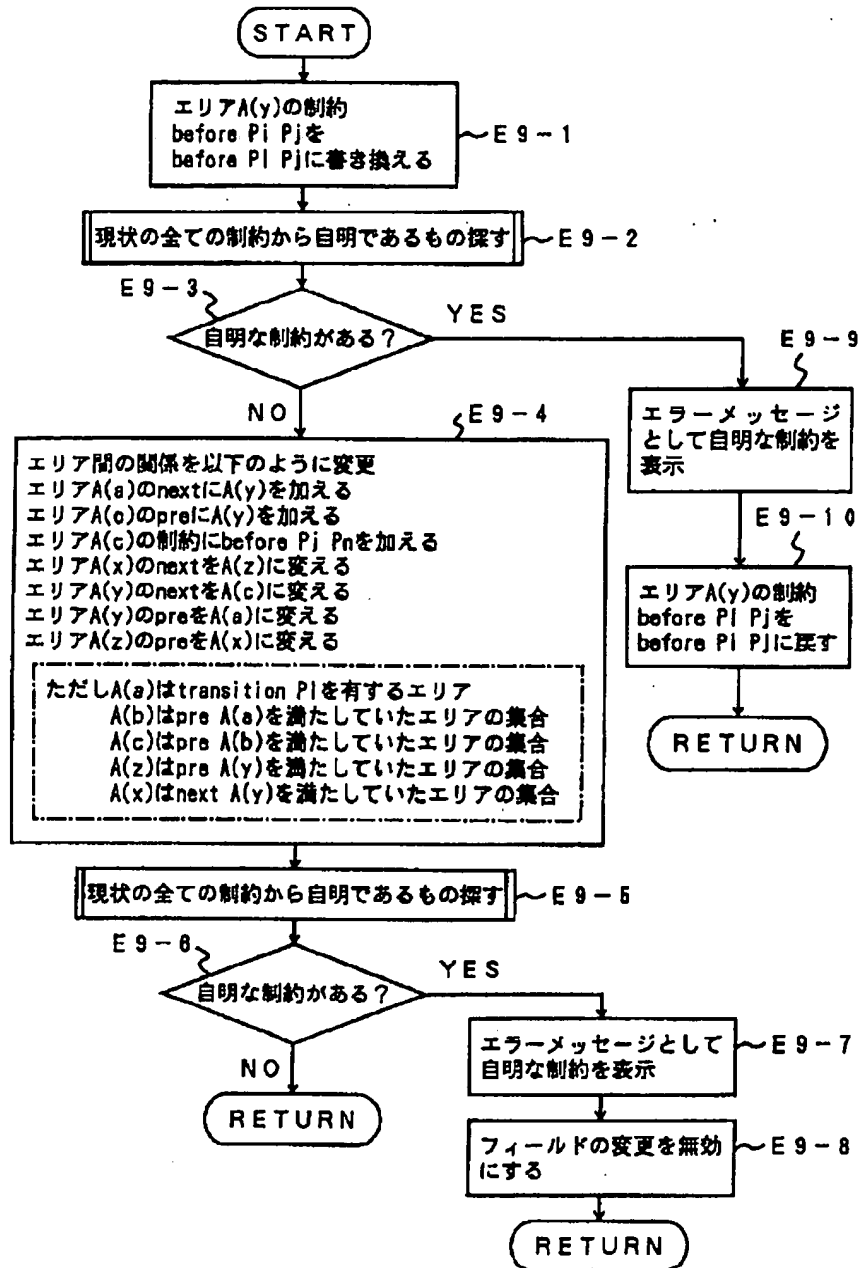
【図45】



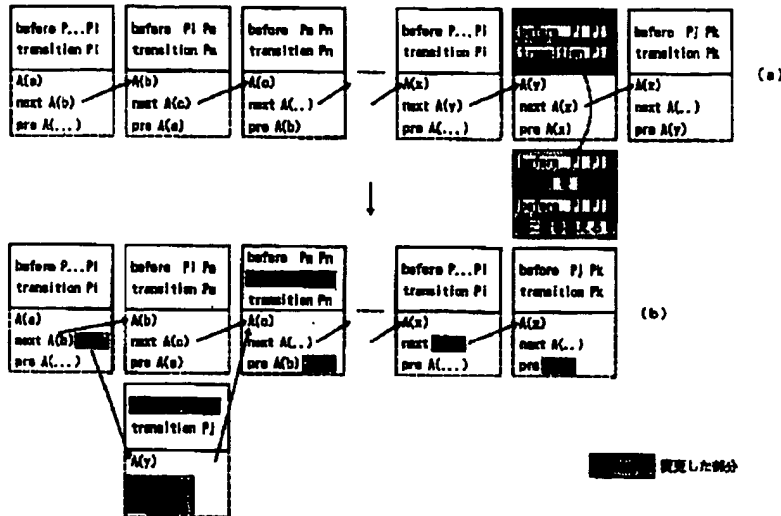
【図46】



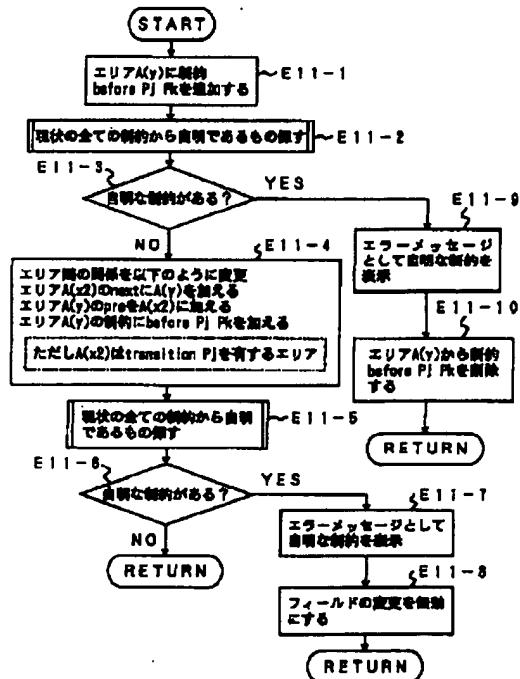
【図47】



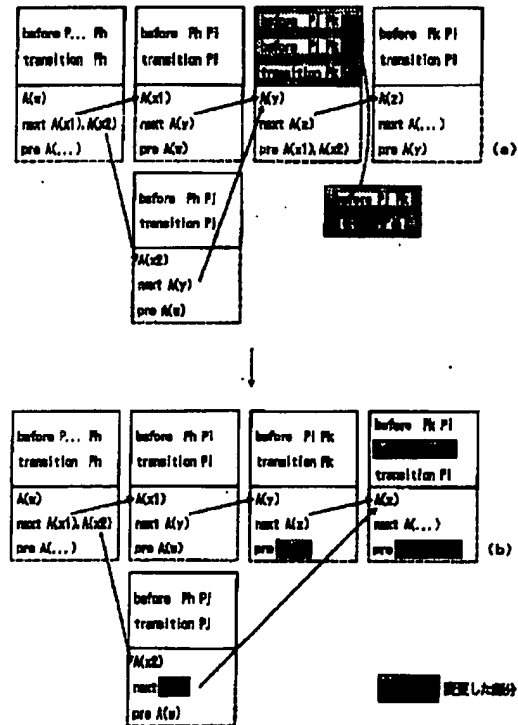
【図48】



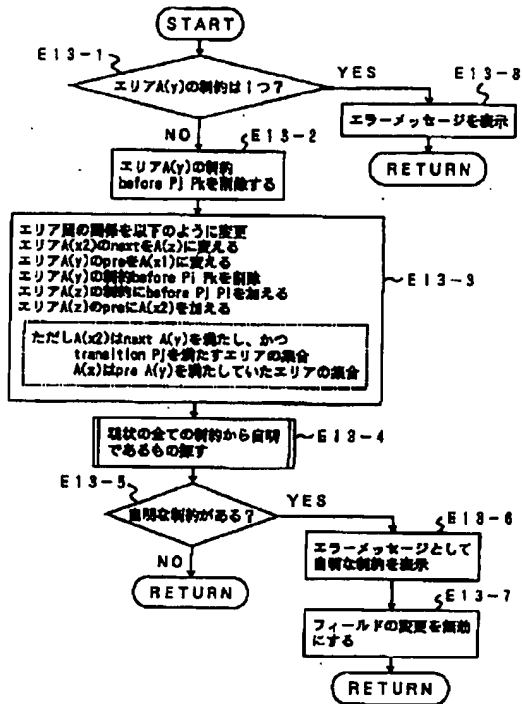
【図49】



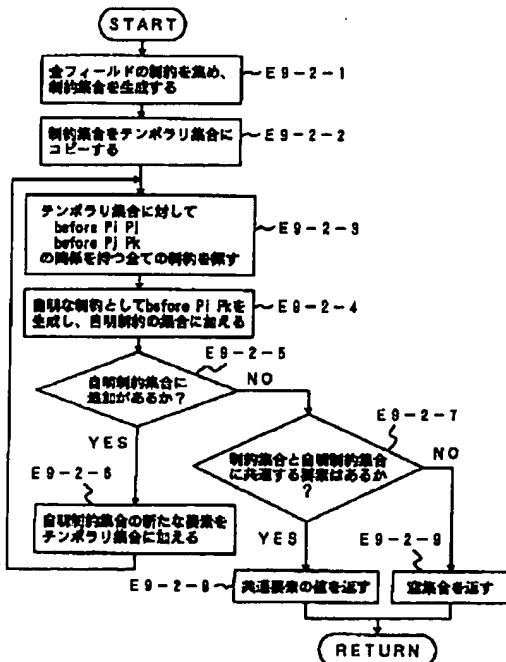
【図50】



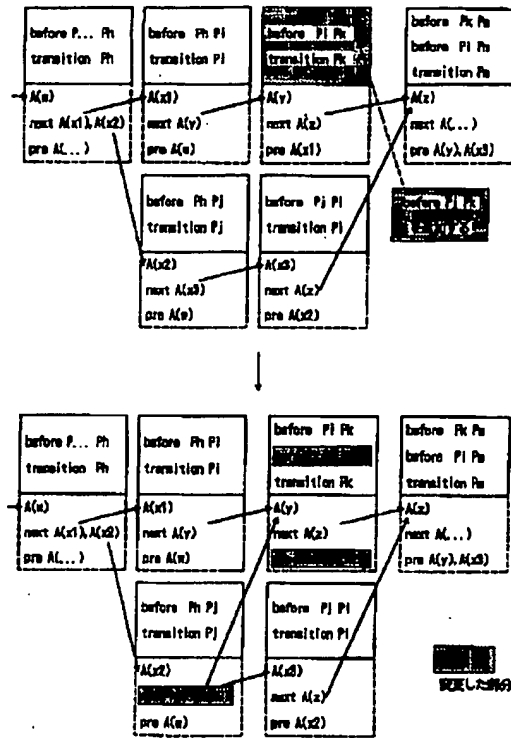
【図51】



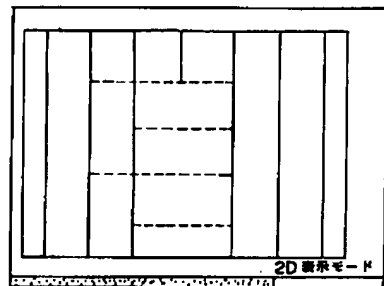
【図53】



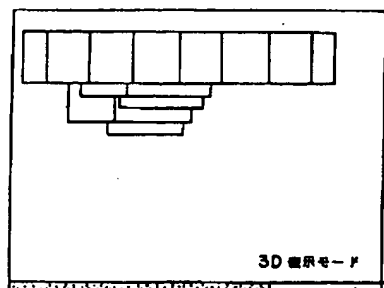
【図52】



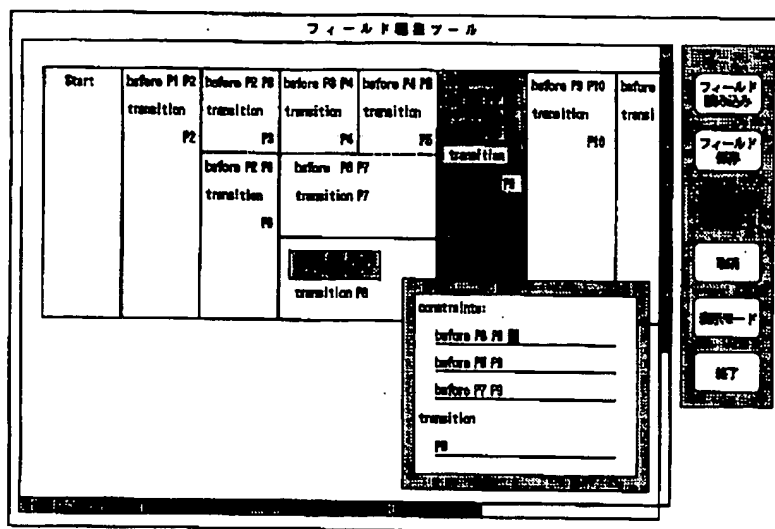
【図57】



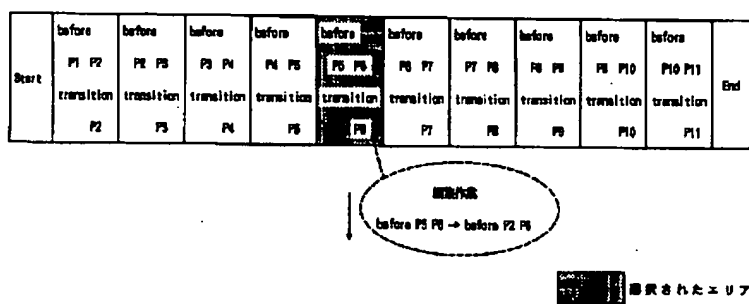
【図58】



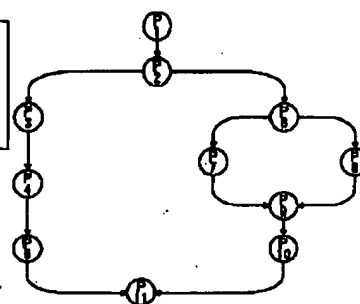
【图 5-6】



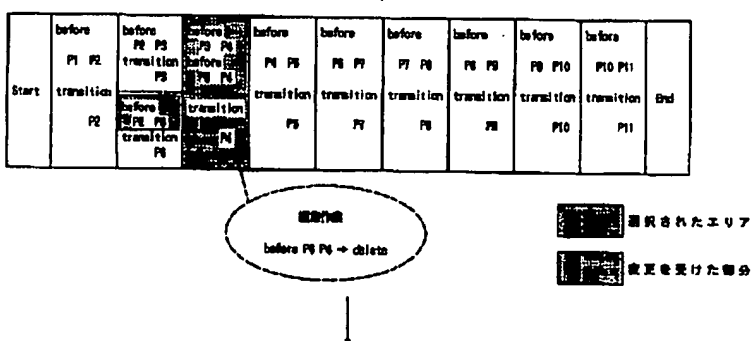
【图 59】



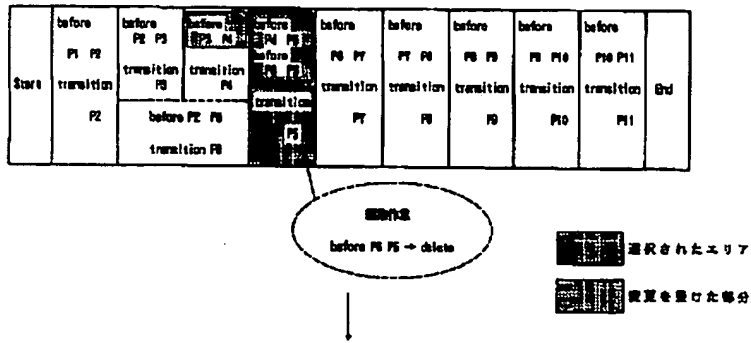
【圖 69】



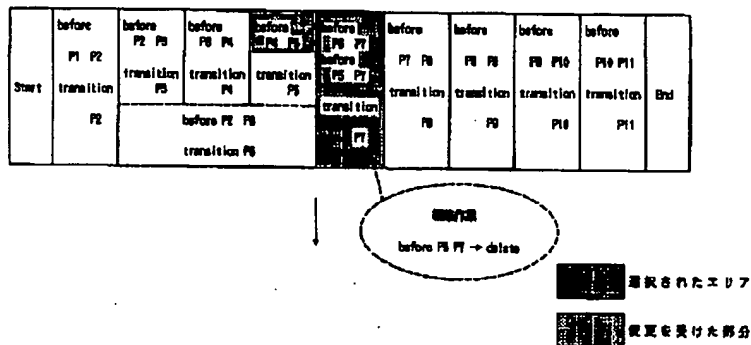
【圖 60】



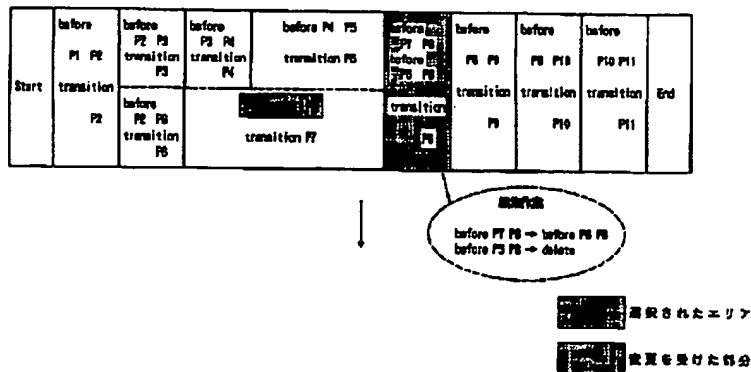
【図 6 1】



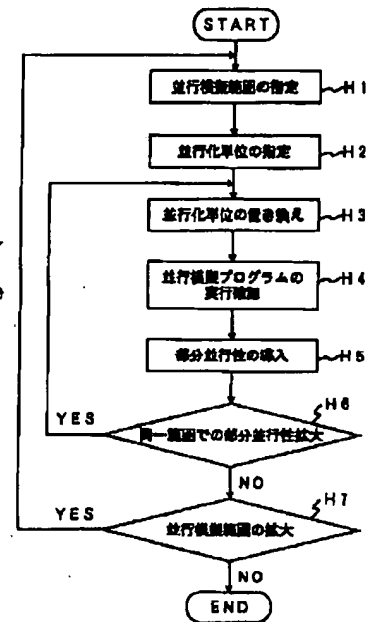
【図 6 2】



【図 6 3】



【図 7 4】

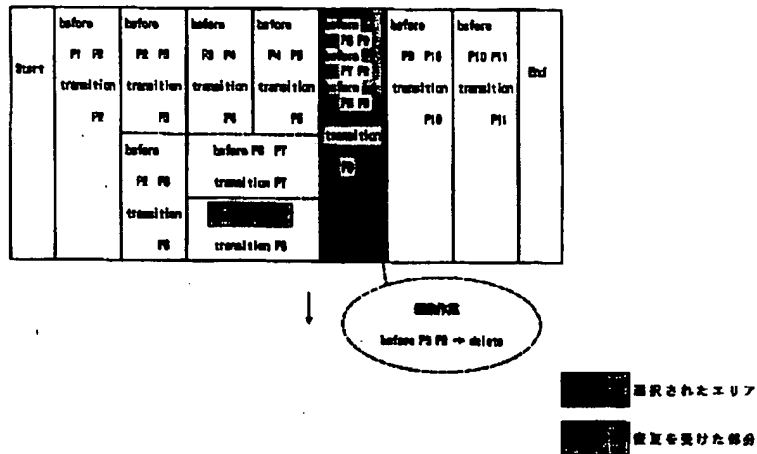


【図 7 3】

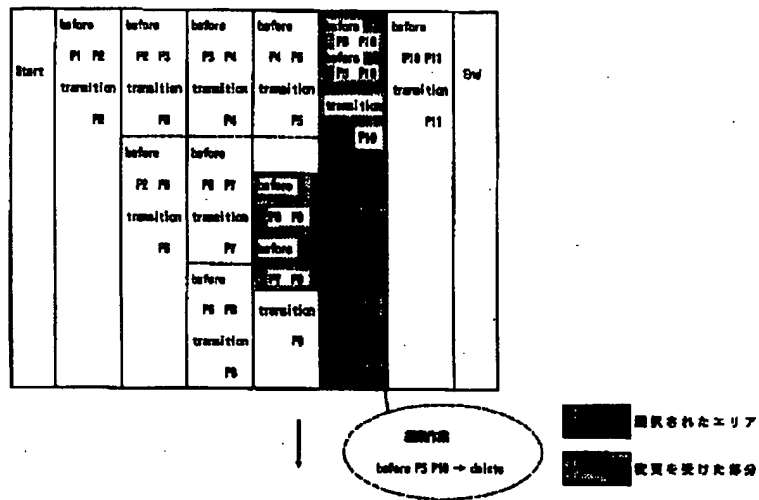




【図64】



【図65】



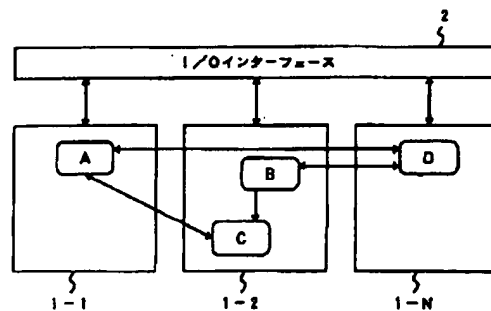
【図68】

```

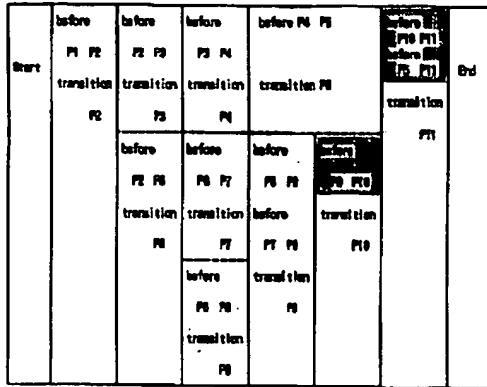
main()
{
  procedure P1;
  procedure P2;
  par
    procedure P3;
    procedure P4;
    procedure P5;
    procedure P6;
    procedure P7;
    procedure P8;
    procedure P9;
    procedure P10;
    procedure P11;
  end
}

```

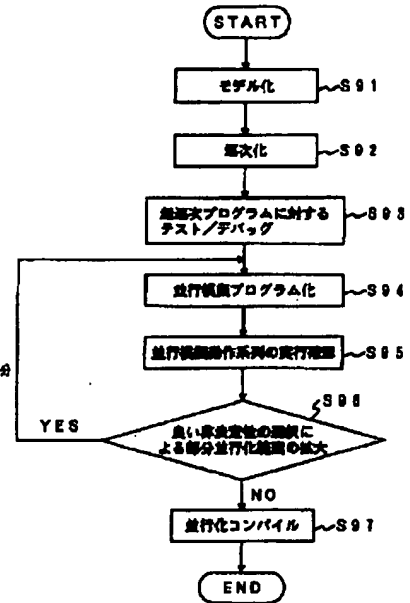
【図79】



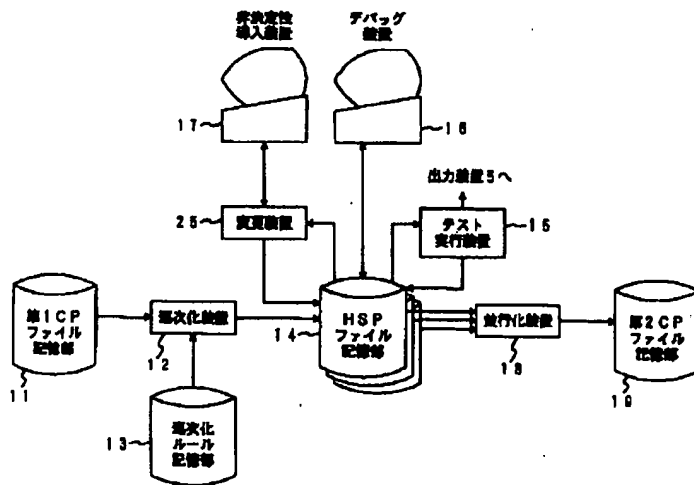
【図66】



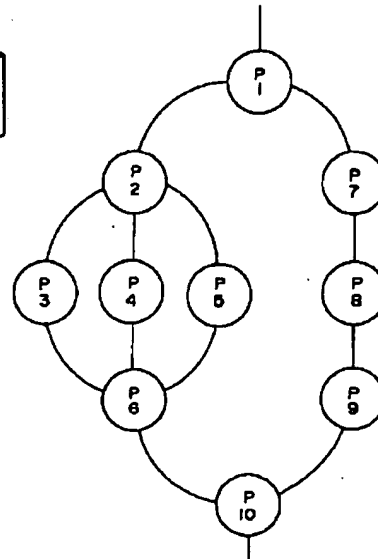
【図71】



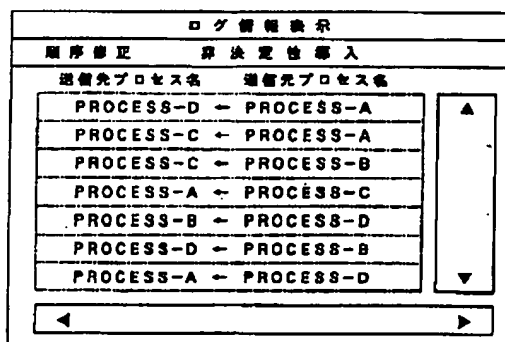
【図70】



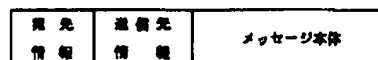
【図72】



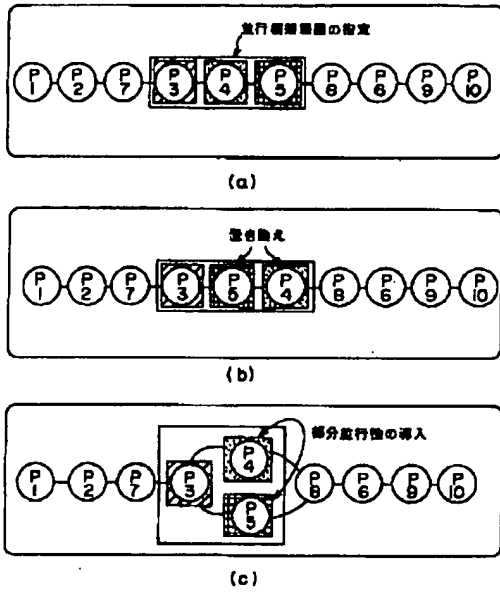
【図82】



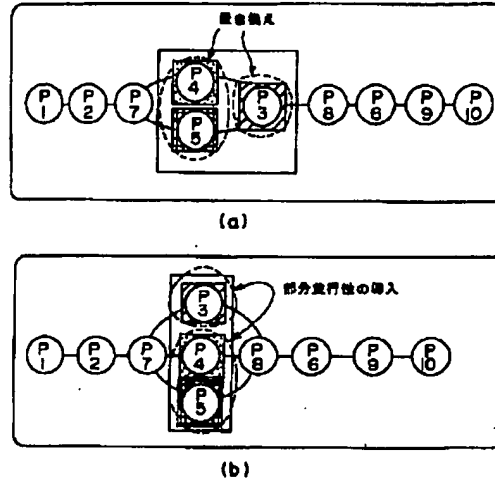
【図84】



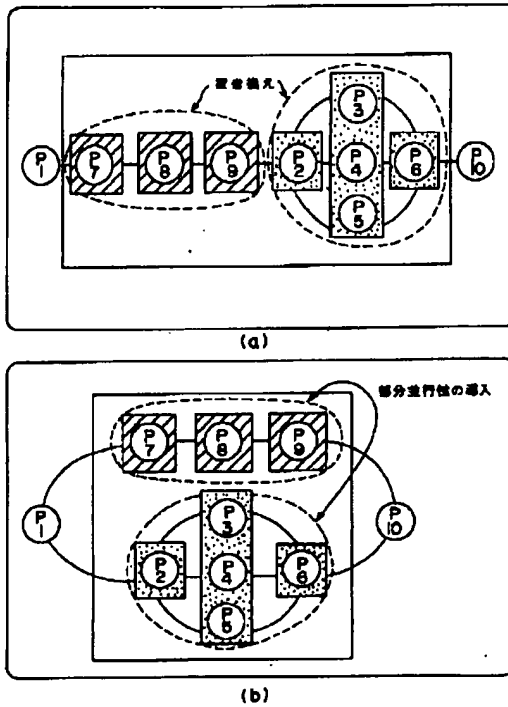
【図 75】



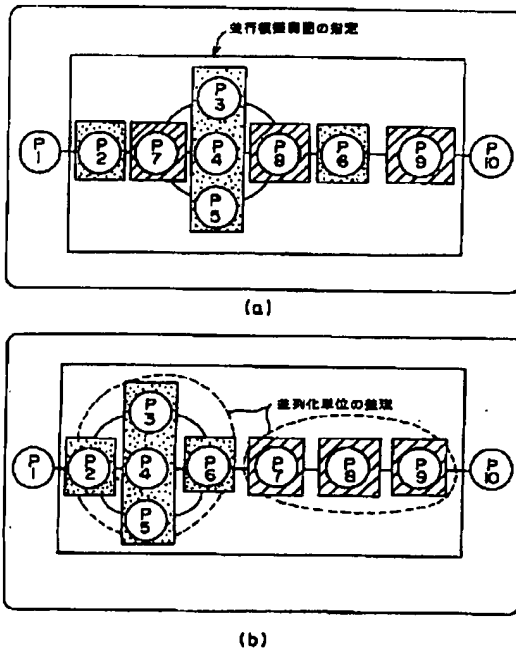
【図 76】



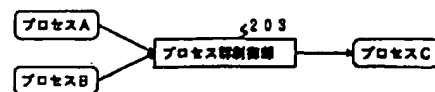
【図 78】



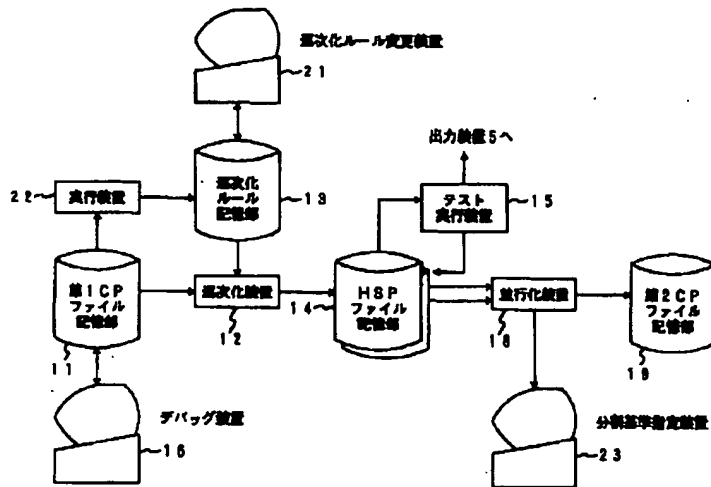
【図 77】



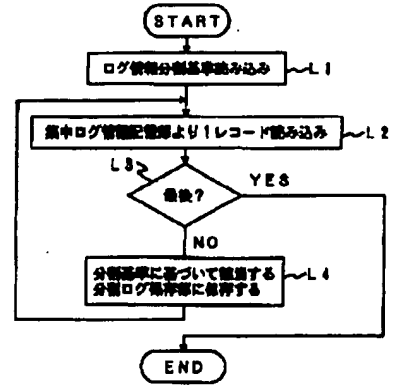
【図 85】



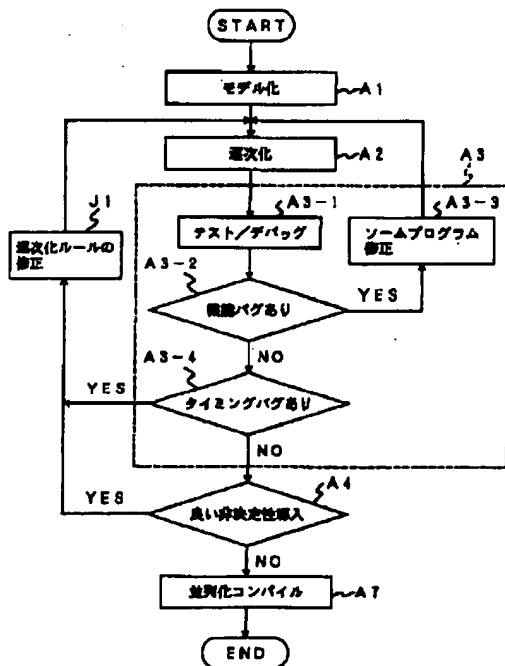
【図80】



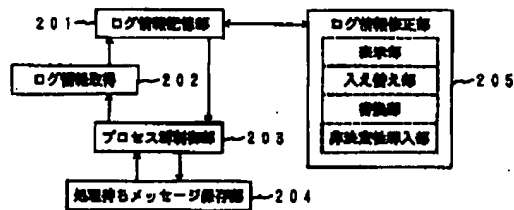
【図95】



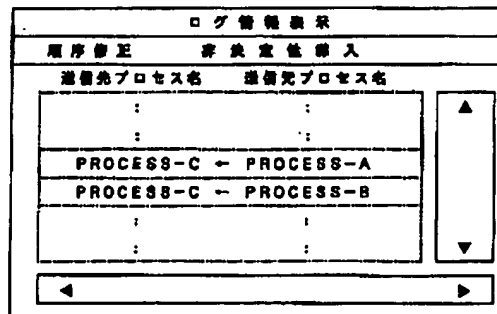
【図81】



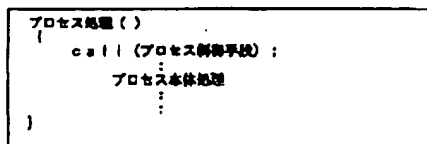
【図83】



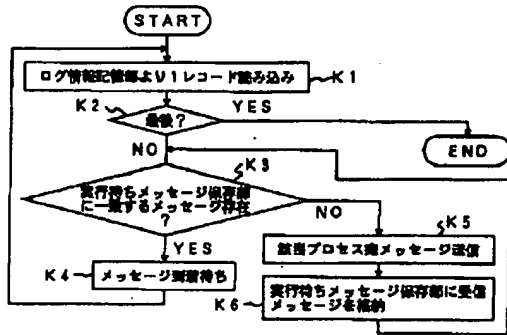
【図86】



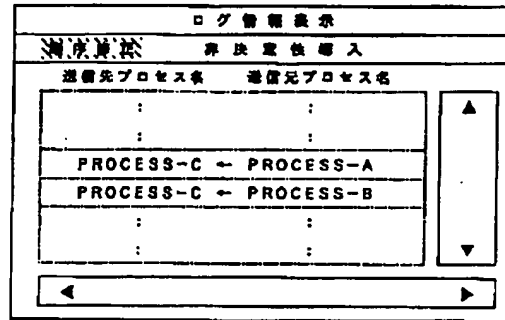
【図97】



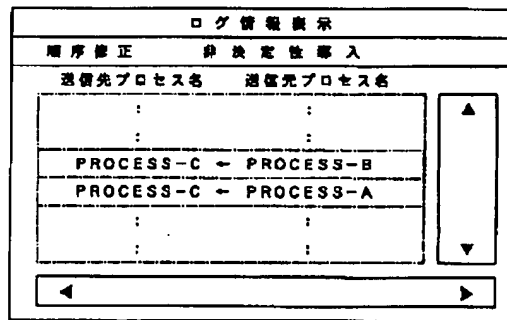
【図87】



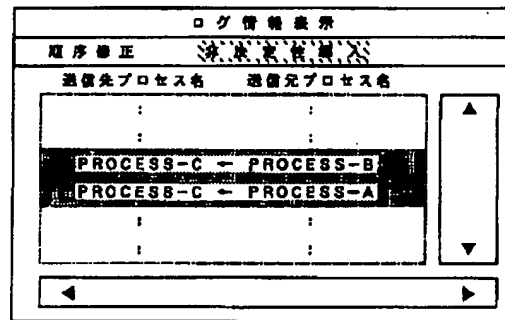
【図88】



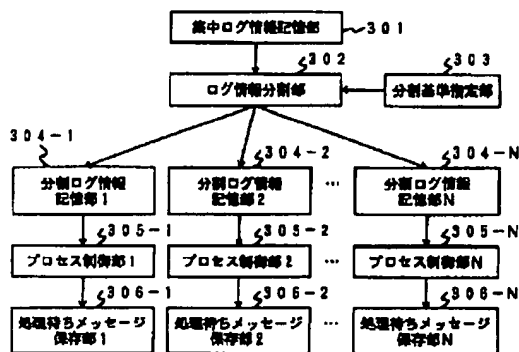
【図89】



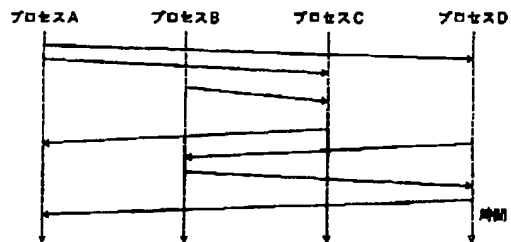
【図90】



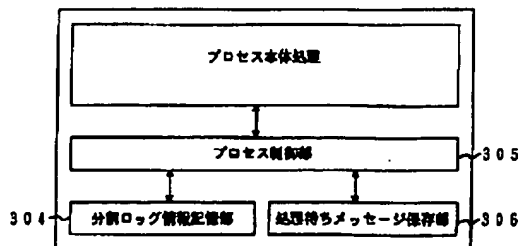
【図92】



【図93】



【図99】



【図106】

セクションの管理テーブルを指すポインタ
出口1が接続される部品テーブルを指すポインタ
出口2が接続される部品テーブルを指すポインタ

【図91】

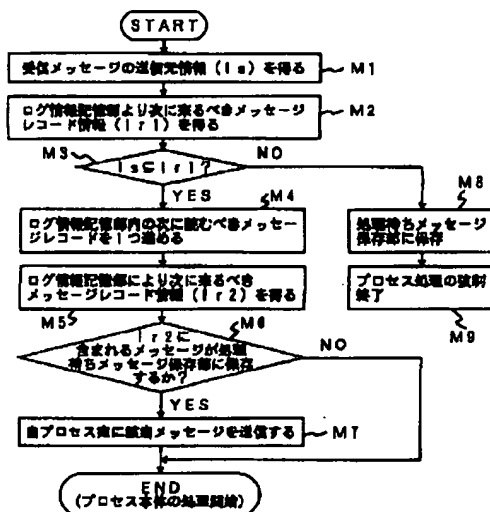
ログ情報表示	
順序修正	非決定性導入
送信元プロセス名	送信先プロセス名
:	:
:	:
PROCESS-C ← PROCESS-B	
PROCESS-C ← PROCESS-A	
:	:
:	:

(a)

ログ情報表示	
順序修正	非決定性導入
送信元プロセス名	送信先プロセス名
:	:
:	:
PROCESS-C ← PROCESS-A	
PROCESS-C ← PROCESS-B	
:	:
:	:

(b)

【図98】



【図94】

ログ情報表示			
送信元プロセス名	送信先プロセス名	送信元プロセス名	送信先プロセス名
PROCESS-D ← PROCESS-A			
PROCESS-C ← PROCESS-A		PROCESS-C ← PROCESS-B	
PROCESS-C ← PROCESS-B		PROCESS-C ← PROCESS-A	
PROCESS-A ← PROCESS-C			
PROCESS-B ← PROCESS-D			
PROCESS-D ← PROCESS-B			
PROCESS-A ← PROCESS-D			

【図96】

プロセスA用分割ログ情報 (分割ログ情報記憶部A)

プロセスA用分割ログ情報表示			
送信元プロセス名	送信先プロセス名	送信元プロセス名	送信先プロセス名
PROCESS-A ← PROCESS-C			
PROCESS-A ← PROCESS-D			

(a)

プロセスB用分割ログ情報 (分割ログ情報記憶部B)

プロセスB用分割ログ情報表示			
送信元プロセス名	送信先プロセス名	送信元プロセス名	送信先プロセス名
PROCESS-B ← PROCESS-D			

(b)

プロセスC用分割ログ情報 (分割ログ情報記憶部C)

プロセスC用分割ログ情報表示			
送信元プロセス名	送信先プロセス名	送信元プロセス名	送信先プロセス名
PROCESS-C ← PROCESS-A		PROCESS-C ← PROCESS-B	
PROCESS-C ← PROCESS-B		PROCESS-C ← PROCESS-A	

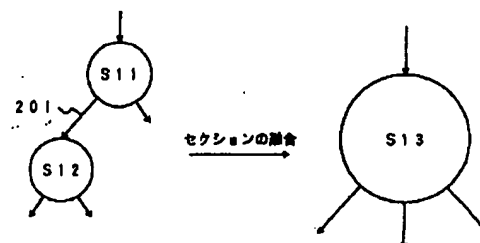
(c)

プロセスD用分割ログ情報 (分割ログ情報記憶部D)

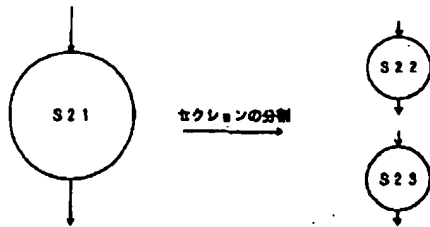
プロセスD用分割ログ情報表示			
送信元プロセス名	送信先プロセス名	送信元プロセス名	送信先プロセス名
PROCESS-D ← PROCESS-A			
PROCESS-D ← PROCESS-B			

(d)

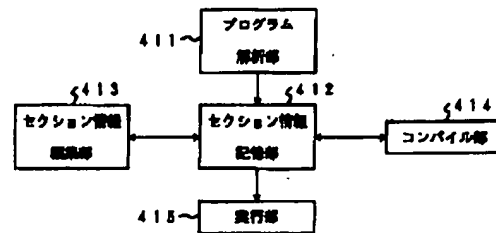
【図101】



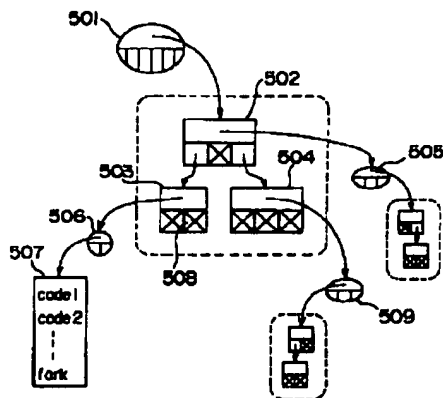
【図102】



【図103】



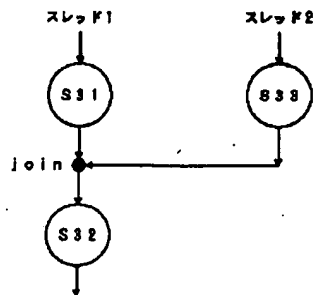
【図104】



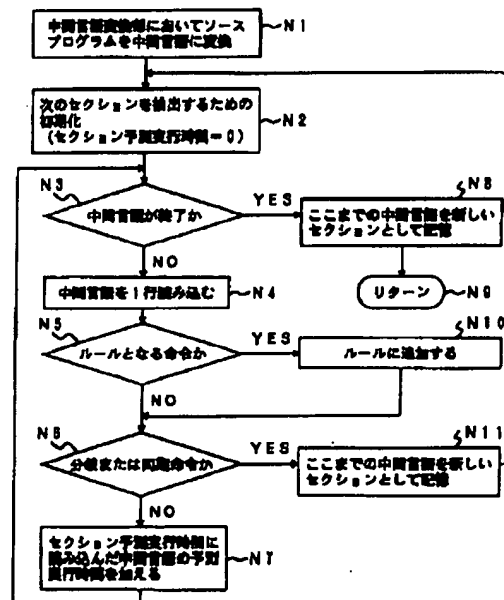
【図105】

ラベル
コンパイルされているかを表すフラグ
コンパイルされたオブジェクトへのポインタ
セクションがプログラムコードを表すフラグ
部品テーブルがプログラムコードを表すポインタ
出口1を指すポインタ
出口1までの予想実行時間
出口2を指すポインタ
出口2までの予想実行時間

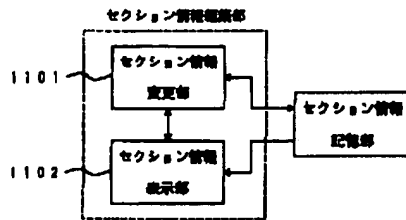
【図107】



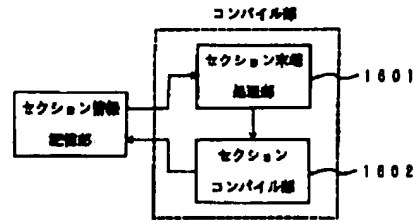
【図109】



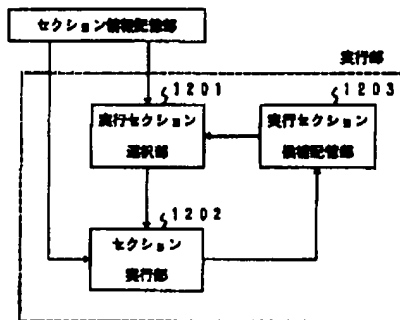
【図110】



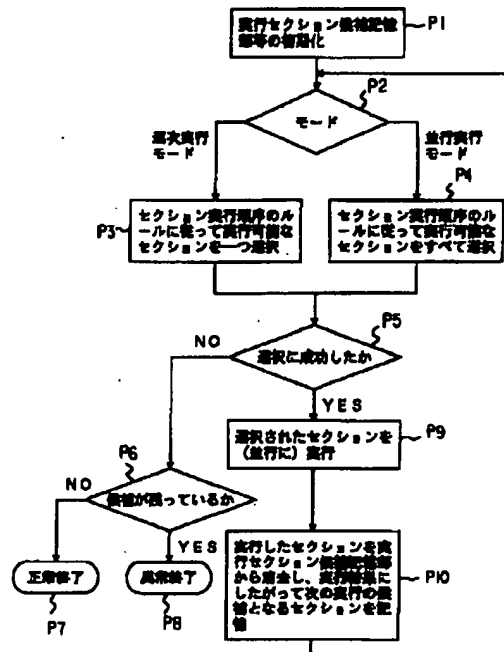
【図111】



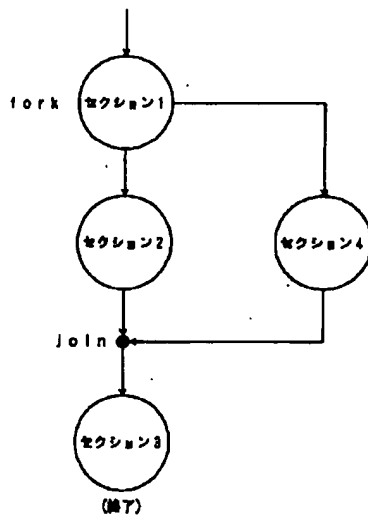
【図112】



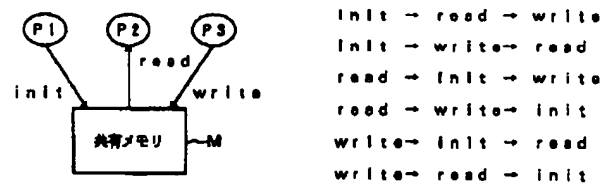
【図113】



【図114】

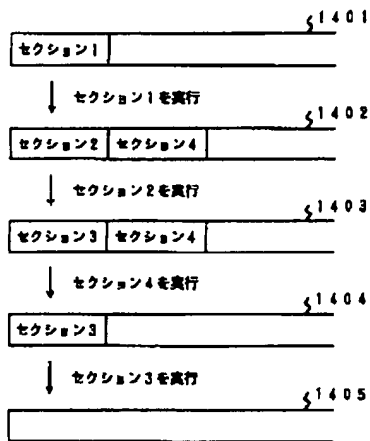


【図116】

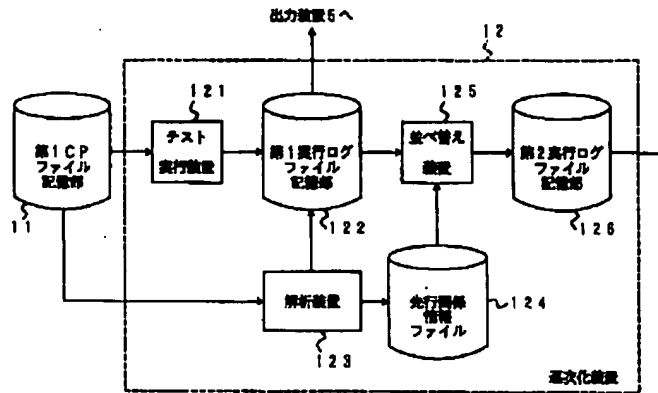




【図115】



【図117】



【図119】

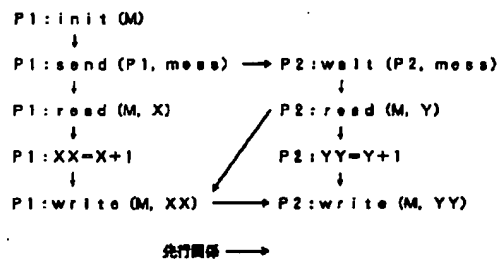
```

CP=P1|P2
P1:
begin
  extern int M;
  int X, XX;
  init (M);
  send (P2, mess);
  read (M, X);
  XX=X+1;
end

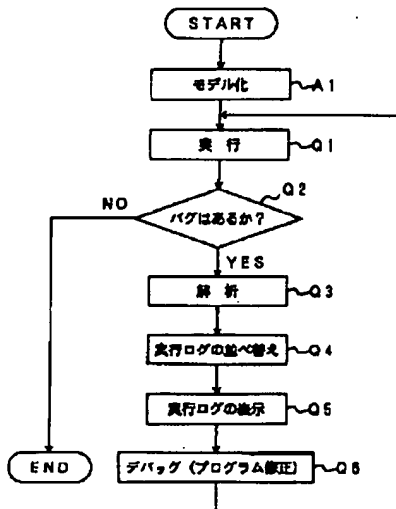
P2:
begin
  extern int M;
  int Y, YY;
  wait (P1, mess);
  read (M, Y);
  YY=Y-1;
  write (M, YY);
end

```

【図121】



【図118】



【図120】

```

実行ログ:
P1: init (M);
P1: send (P2, mess);
P1: read (M, X);
P2: wait (P1, mess);
P1: XX=X+1;
P2: read (M, Y);
P2: YY=Y-1;
P1: write (M, XX);
P2: write (M, YY);

```

【図122】

並び替えた実行ログ:  
 P1: init (M);  
 P1: send (P2, mess);  
 P1: read (M, X);  
 P1: XX=X+1;  
 P2: wait (P1, mess);  
 P2: read (M, Y);  
 P1: write (M, XX);  
 P2: YY=Y-1;  
 P2: write (M, YY);

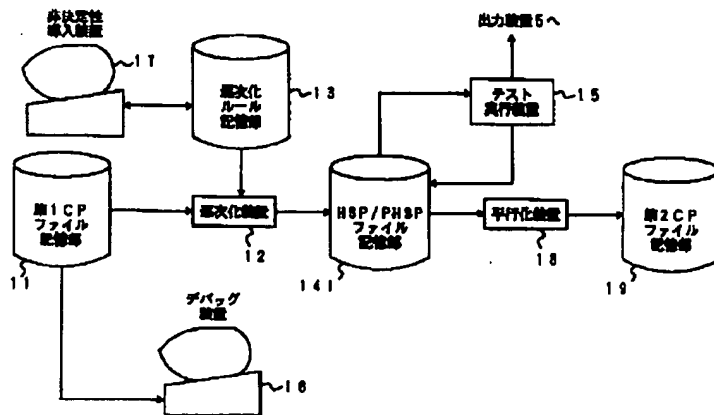
【図123】

```

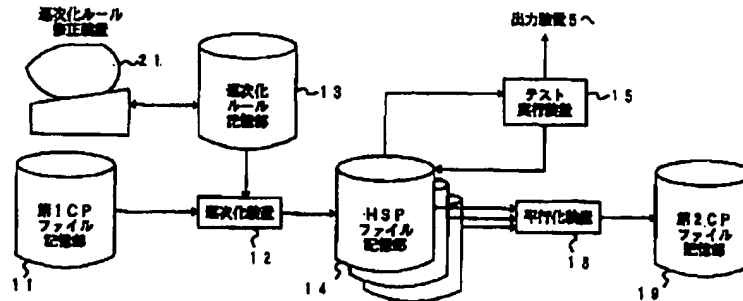
CP=P1||P2
P1:
begin
extern int M;
int X, XX;
init (M);
read (M, X);
XX=X+1;
write (M, XX);
send (P2, mess);
end

P2:
begin
extern int M;
int Y, YY;
wait (P1, mess);
read (M, X);
YY=Y-1;
write (M, YY);
end
  
```

【図124】



【図126】



フロントページの続き

(72)発明者 關 俊文  
 神奈川県川崎市幸区柳町70番地 株式会社  
 東芝柳町工場内  
 (72)発明者 永井 保夫  
 神奈川県川崎市幸区柳町70番地 株式会社  
 東芝柳町工場内  
 (72)発明者 半田 恵一  
 神奈川県川崎市幸区柳町70番地 株式会社  
 東芝柳町工場内

(72)発明者 伊藤 聡  
 神奈川県川崎市幸区柳町70番地 株式会社  
 東芝柳町工場内  
 (72)発明者 澤島 信介  
 神奈川県川崎市幸区小向東芝町1番地 株  
 式会社東芝研究開発センター内  
 (72)発明者 田原 康之  
 神奈川県川崎市幸区柳町70番地 株式会社  
 東芝柳町工場内  
 (72)発明者 塩谷 英明  
 神奈川県川崎市幸区柳町70番地 株式会社  
 東芝柳町工場内

**This Page is Inserted by IFW Indexing and Scanning  
Operations and is not part of the Official Record**

**BEST AVAILABLE IMAGES**

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☒ **BLACK BORDERS**
- ☐ **IMAGE CUT OFF AT TOP, BOTTOM OR SIDES**
- ☒ **FADED TEXT OR DRAWING**
- ☒ **BLURRED OR ILLEGIBLE TEXT OR DRAWING**
- ☐ **SKEWED/SLANTED IMAGES**
- ☐ **COLOR OR BLACK AND WHITE PHOTOGRAPHS**
- ☐ **GRAY SCALE DOCUMENTS**
- ☐ **LINES OR MARKS ON ORIGINAL DOCUMENT**
- ☐ **REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY**
- ☐ **OTHER:** \_\_\_\_\_

**IMAGES ARE BEST AVAILABLE COPY.**

**As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.**